Assignment Day 11

Task 1:

Explain the below concepts with an example in brief.

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

Answers:

- Nosql Databases

Ans-

A NoSQL database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases.

An approach to database design that can accommodate 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 database provides ability to process very large volumes of data and quickly distribute that data across those distinct computing clusters.

Features: -

•**Generic Data Model** -Heterogeneous containers, including sets, maps, and arrays

•**Dynamic type discovery** and **conversion** –NoSQL analytics systems support runtime type identification and conversion so that custom business logic can be used to dictate analytic treatment of variation.

•**Non-relational** and **De-normalised**- Data is stored in single tables as compared to joining multiple tables.

•**Commodity hardware** - Adding more of the economical servers allows NoSQL databases to scale to handle more data.

•**Highly distributable** - Distributed databases can store and process a set of information on more than one device.

Examples of **NoSQL** are: Cassandra, MongoDB, HBase, Redis etc.

● Types of Nosql Databases

Ans-

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. Wide-Column store NoSQL database
4. Graph stores NoSQL database

Each can be explained in brief as follows:

a>Key-value stores

Key-value stores, or key-value databases, implement a simple data model that pairs a unique key with an associated value. Because this model is simple, it can lead to the development of key-value databases, which are extremely performant and highly scalable for session management and caching in web applications. Implementations differ in the way they are oriented to work with RAM, solid-state drives or disk drives.

e.g. Riak, MemchacheDB, Aerospike, Berkeley DB and Redis.

b>Document databases

Document databases, also called document stores, store semi-structured data and descriptions of that data in document format. They allow developers to create and update programs without needing to reference master schema. Use of document databases has increased along with use of JavaScript and the JavaScript Object Notation (JSON), a data interchange format that has gained wide currency among web application developers, although XML and other data formats can be used as well. Document databases are used for content management and mobile application data handling.

e.g. Couchbase Server, MarkLogic, CouchDB, Document DB and MongoDB

c>Wide-column stores

Wide-column stores organize data tables as columns instead of as rows. Wide-column stores can be found both in SQL and NoSQL databases. Wide-column stores can query large data volumes faster than conventional relational databases. A wide-column data store can be used for recommendation engines, CATALOGS, fraud detection and other types of data processing.

e.g. HBase, Google Bigtable and Cassandra.

d>Graph stores

Graph data stores organizes data as nodes, which are like records in a relational database, and edges, which represent connections between nodes. Because the graph system stores the relationship between nodes, it can support richer representations of data relationships. Also, unlike relational models reliant on strict schemas, the graph data model can evolve over time and use. Graph databases are applied in systems that must map relationships, such as reservation systems or customer relationship management.

e.g AllegroGraph, Neo4j, IBM Graph and Titan.

● CAP Theorem

Ans-

CAP theorem, also named **Brewer's theorem** after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

In other words, the CAP theorem states that in the presence of a network partition, one must 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.

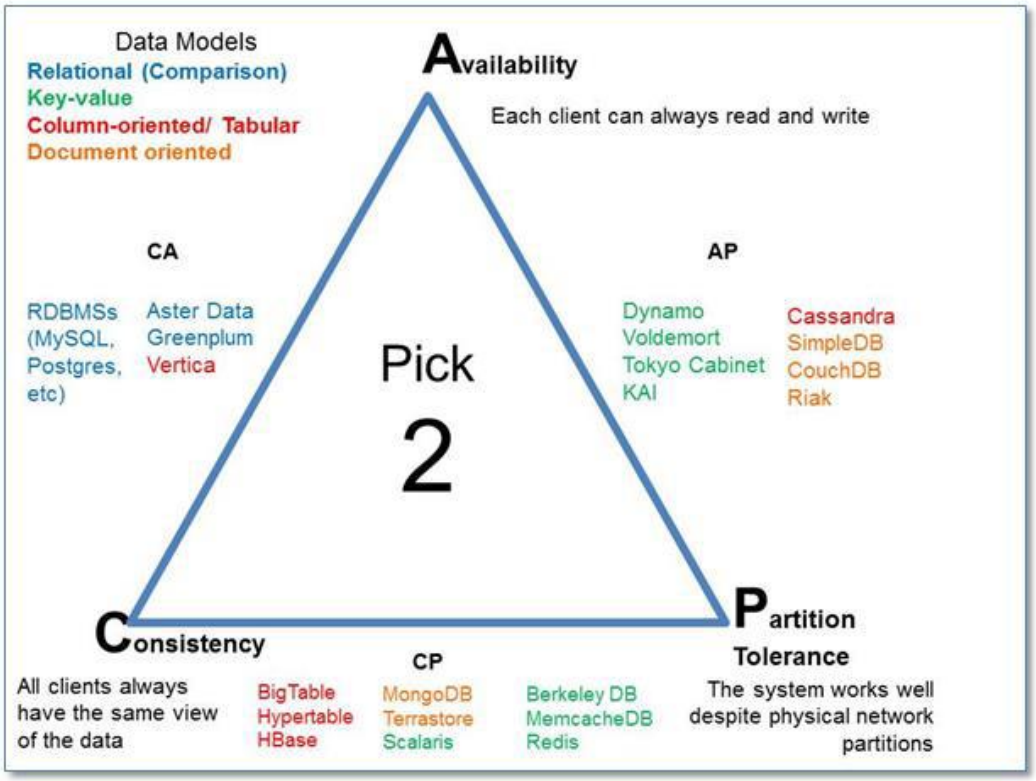
Partition Tolerance: this means the system continues to function even if the communication among the servers are unreliable, i.e., the servers may be partitioned into multiple groups that cannot communicate with one another.

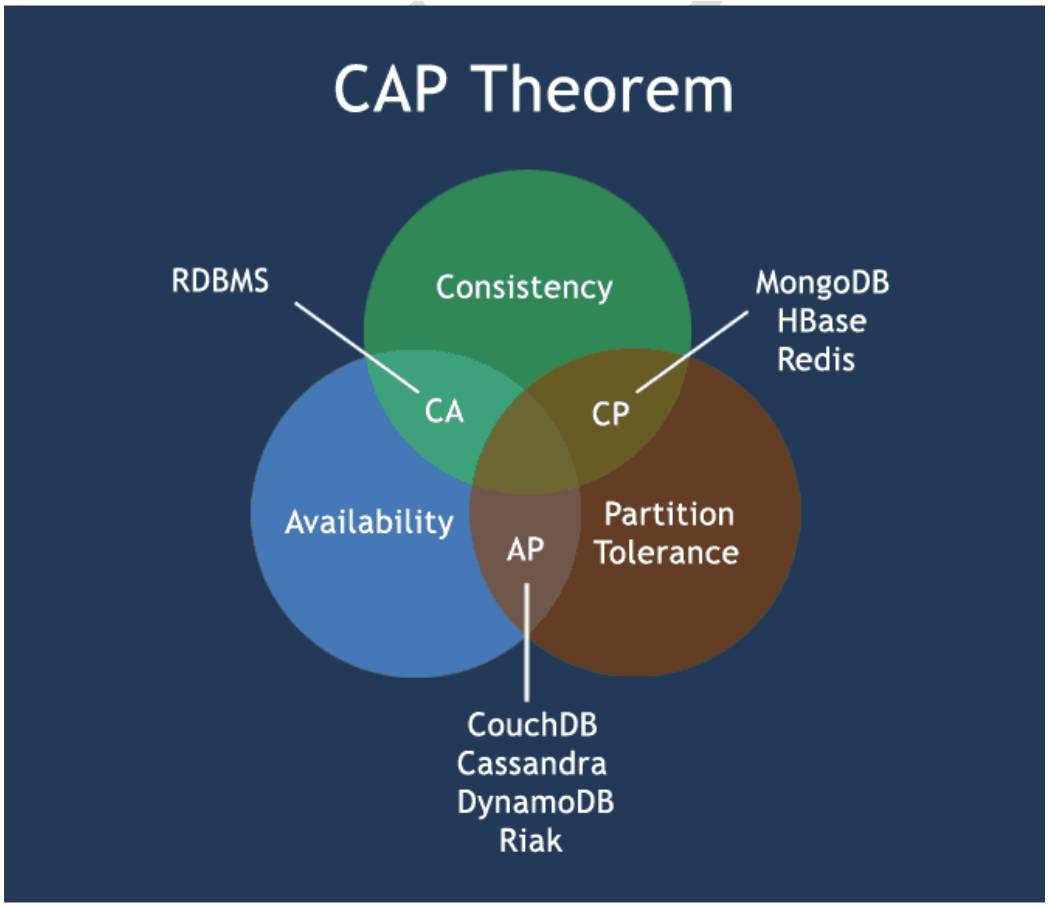
CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore, all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem. Here is the brief description of three combinations CA, CP, AP.

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.





●HBase Architecture

Ans-

HBASE is a distributed column oriented database built on top of hadoop to provide real time access to Big Data. HBase is composed of three types of servers in a master slave type of architecture.

•**Region servers** serve data for reads and writes.

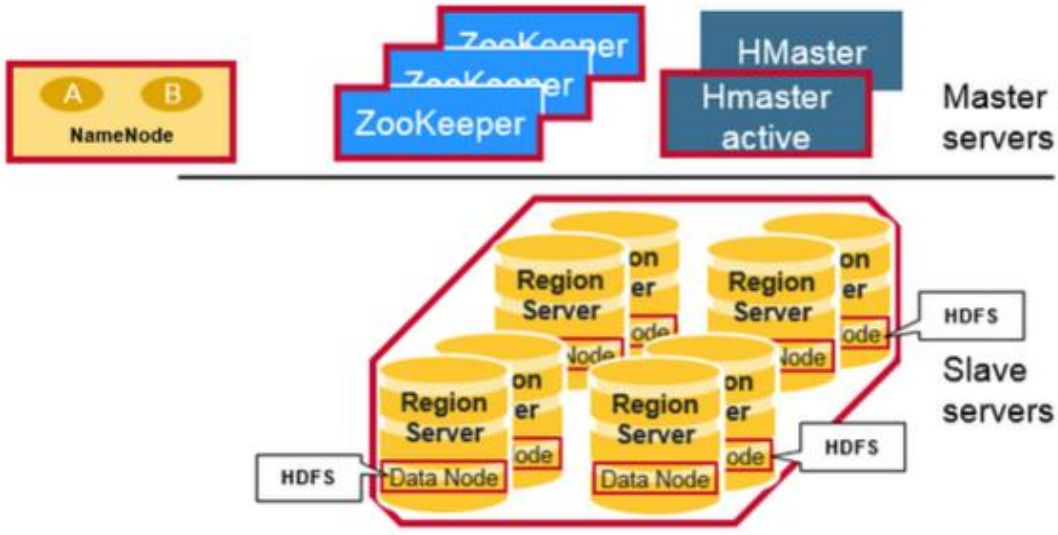
•**HBase Master** process handles the Region assignment, DDL (create, delete tables) operations

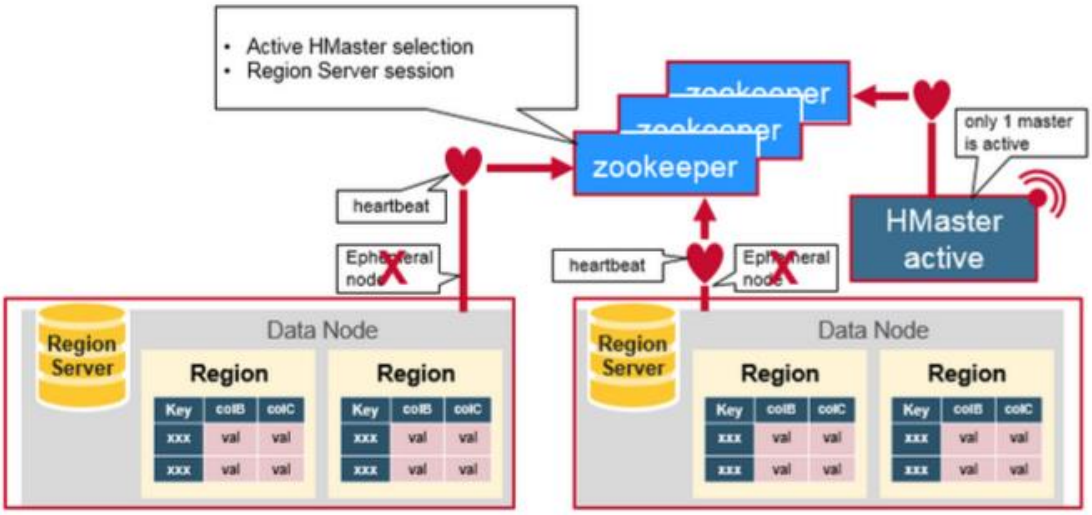
•**Zookeeper** maintains a live cluster state.

•The **Hadoop Data Node** stores the data that the Region Server is managing.

•All HBase data is stored in **HDFS** files.

•The **Name Node** maintains metadata information for all the physical data blocks that comprise the files.





Components all Together:

Regions

HBase Tables are divided horizontally by row key range into "Regions." A region contains all rows in the table between the region's start key and end key. Regions are assigned to the nodes in the cluster, called "Region Servers," and these serve data for reads and writes. A region server can serve about 1,000 regions.

HBase HMaster

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

ZooKeeper: The Coordinator

HBase uses ZooKeeper as a distributed coordination service to maintain server state in the cluster. Zookeeper maintains which servers are alive and available, and provides server failure notification. Zookeeper uses consensus to guarantee common shared state. Note that there should be three or five machines for consensus.

HBase Meta Table

This META table is an HBase table that keeps a list of all regions in the system.

The .META. table is like a B tree.

The .META. table structure is as follows:

- Key: region start key,region id

- Values: RegionServer

Region Server Components

A 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: is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.

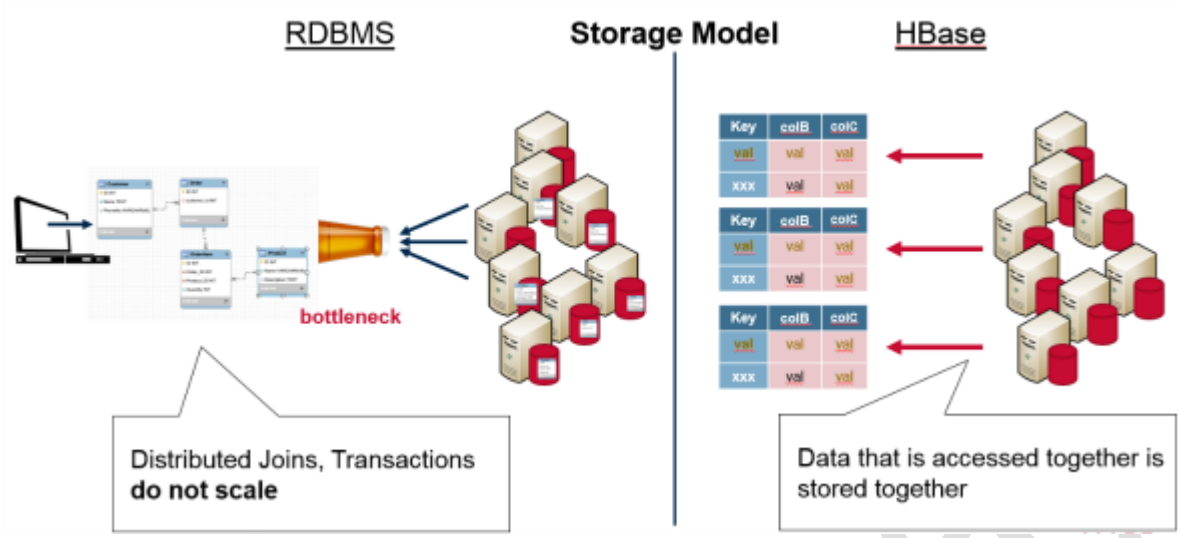
MemStore: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.

Hfiles store the rows as sorted KeyValues on disk.

● HBase vs RDBMS

Ans-

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	Uses SQL
Wide tables	Narrow Tables
Joins using MR – not optimized	optimized for Joins(small, fast ones)
Tight – Integration with MR	Not really



Task 2:

Execute blog present in below link

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

Ans:

ImportTsv takes data from HDFS into HBase via Puts. Find below the syntax used to load data via Puts (i.e., non-bulk loading):

Step1:
create 'bulktable', 'cf1', 'cf2'

Step 2:

mkdir hbase

cd hbase

pwd

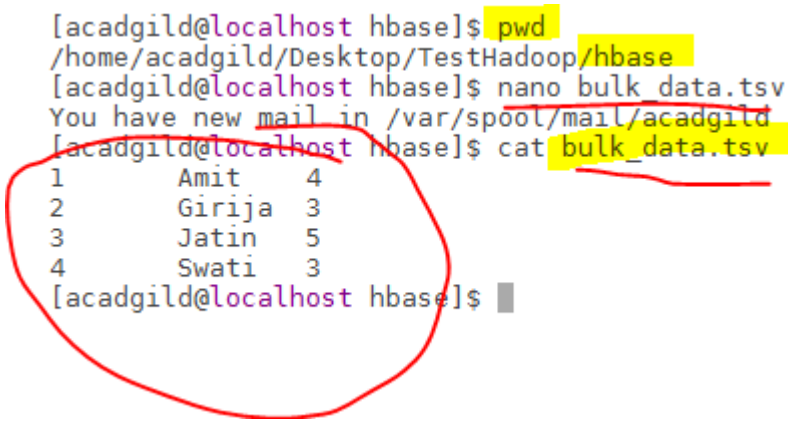
/home/acadgild/Desktop/TestHadoop/hbase

Step 3:

nano bulk_data.tsv

cat bulk_data.tsv

ScreenShot:



Step 4:

hadoop fs -put bulk_data.tsv /hbase/

(puts file into HDFS)

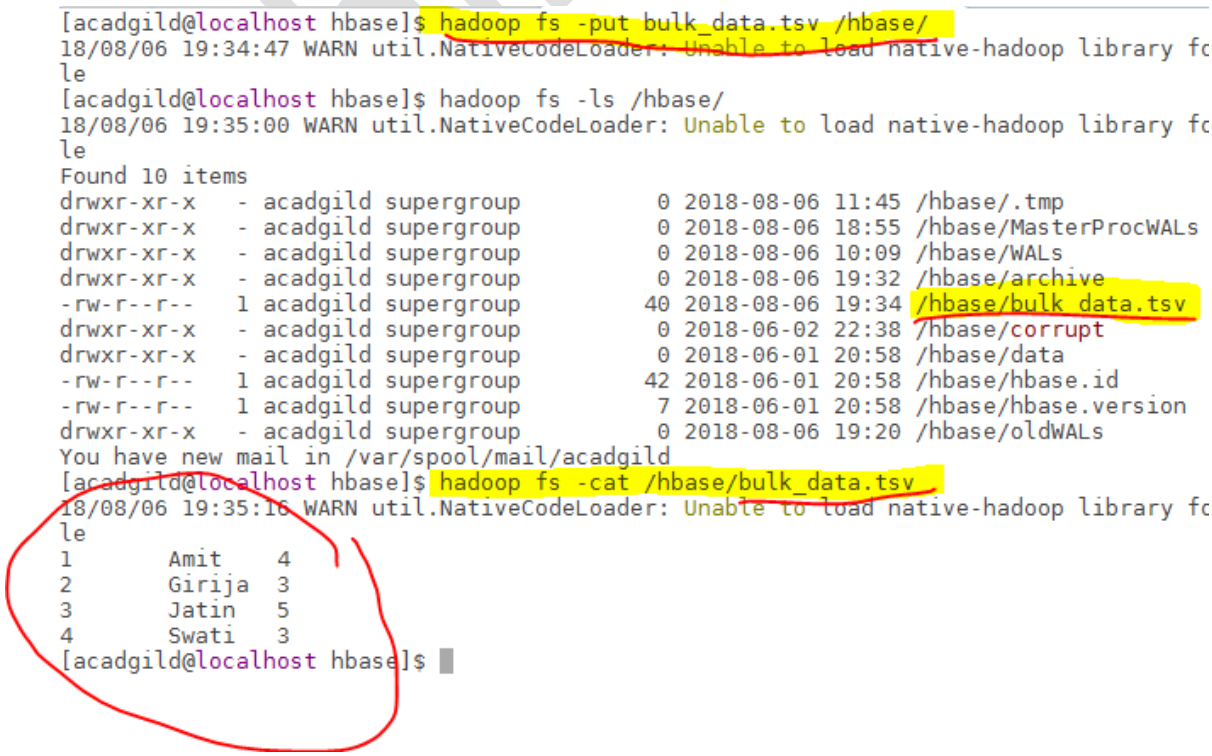
hadoop fs -ls /hbase/

(lists files present in /hbase/)

hadoop fs -cat /hbase/bulk_data.tsv

(Displays the content of /hbase/bulk_data.tsv)

ScreenShot:



Step 5:

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>

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

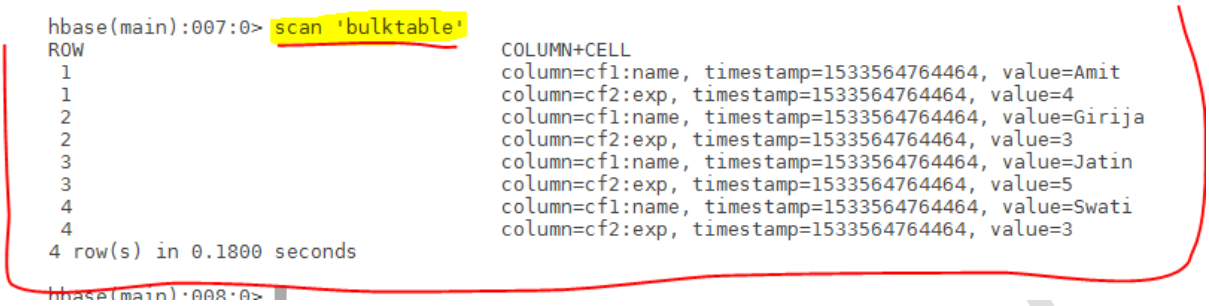
ScreenShot:

```
[acadgild@localhost hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase
/bulk_data.tsv
2018-08-06 19:42:44,548 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-1.2.6/lib/slf4j-log4j12-1.7.5.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/Stat
icLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2018-08-06 19:42:45,569 INFO  [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to ZooKeeper ensemble=l
ocalhost:2181
2018-08-06 19:42:45,596 INFO  [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT
2018-08-06 19:42:45,596 INFO  [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-08-06 19:42:45,596 INFO  [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-08-06 19:42:45,596 INFO  [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-08-06 19:42:45,597 INFO  [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-08-06 19:42:45,597 INFO  [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hbase-1.2.6/conf:/usr/j
ava/jdk1.8.0_151/lib/tools.jar:/home/acadgild/install/hbase/hbase-1.2.6:/home/acadgild/install/hbase/hbase-1.2.6/lib/activation-1.1.jar:/home/aca
dgild/install/hbase/hbase-1.2.6/lib/aopalliance-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-i18n-2.0.0-M15.jar:/home/acadgild/i
nstall/hbase/hbase-1.2.6/lib/apacheds-kerberos-codec-2.0.0-M15.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/api-asn1-api-1.0.0-M20.jar:/home/
acadgild/install/hbase/hbase-1.2.6/lib/api-util-1.0.0-M20.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/asm-3.1.jar:/home/acadgild/install/hba
se/hbase-1.2.6/lib/avro-1.7.4.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-beanutils-1.7.0.jar:/home/acadgild/install/hbase/hbase-1.2
.6/lib/commons-beanutils-core-1.8.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-cli-1.2.jar:/home/acadgild/install/hbase/hbase-1.2.6
/lib/commons-codec-1.9.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-collections-3.2.2.jar:/home/acadgild/install/hbase/hbase-1.2.6/li
b/commons-compress-1.4.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-configuration-1.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/
lib/commons-daemon-1.0.13.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-digester-1.8.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/
commons-el-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-httpclient-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-i
o-2.4.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-lang-2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-logging-1.2.jar:
/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-math-2.2.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-math3-3.1.1.jar:/home/acad
gild/install/hbase/hbase-1.2.6/lib/commons-net-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/disruptor-3.3.0.jar:/home/acadgild/install/hb
ase/hbase-1.2.6/lib/findbugs-annotations-1.3.9-1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/guava-12.0.1.jar:/home/acadgild/install/hbase/h
base-1.2.6/lib/guice-3.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/guice-servlet-3.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hado
op-annotations-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-auth-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-c
lient-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-common-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-hdfs-2.5
.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-mapreduce-client-app-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-map
reduce-client-common-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-mapreduce-client-core-2.5.1.jar:/home/acadgild/install/hbase/h
```

```
2018-08-06 19:43:13,895 INFO  [main] mapreduce.Job:  map 0% reduce 0%
2018-08-06 19:43:28,036 INFO  [main] mapreduce.Job:  map 100% reduce 0%
2018-08-06 19:43:29,080 INFO  [main] mapreduce.Job: Job job_1533488475608_0001 completed successfully
2018-08-06 19:43:29,415 INFO  [main] mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=139487
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=146
                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)=10354
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=10354
                Total vcore-seconds taken by all map tasks=10354
                Total megabyte-seconds taken by all map tasks=10602496
        Map-Reduce Framework
                Map input records=4
                Map output records=4
                Input split bytes=106
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=138
                CPU time spent (ms)=2500
                Physical memory (bytes) snapshot=125206528
                Virtual memory (bytes) snapshot=2067746816
                Total committed heap usage (bytes)=48758784
        ImportTsv
                Bad Lines=0
```

Step 6:

scan 'bulktable'



Key Notes:

HFiles of data to prepare for a bulk data load, pass the option:

-**Dimporttsv.bulk.output**=/path/for/output
The target table will be created with default column family descriptors if it does not already exist Otheroptions that may be specified with -D include:

-**Dimporttsv.skip.bad.lines**=false – fail if encountering an invalid line

'-**Dimporttsv.separator**=|' – eg separate on pipes instead of tabs
-**Dimporttsv.timestamp**=currentTimeAsLong –use the specified timestamp for the import.
-**Dimporttsv.mapper.class**=my.Mapper – A user-defined Mapper to use instead of org.apache.hadoop.hbase.mapreduce.TsvImporterMapper

********** **End** **********