



# BIG DATA HADOOP & SPARK TRAINING

ASSIGNMENT ON ADVANCE HBASE



Rashmi Krishna

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

## • NoSQL Databases

A NoSQL provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called Not only SQL to emphasize that they may support SQL-like query languages.

Motivations for this approach include:

- simplicity of design
- simpler horizontal scaling to clusters of machines
- finer control over availability.

The data structures used by NoSQL databases e.g. key-value, wide column, graph, or document are different from those used by default in relational databases, make some operations faster in NoSQL. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables. Many NoSQL stores compromise consistency in favor of availability, partition tolerance, and speed.

Barriers to the greater adoption of NoSQL stores include:

1. the use of low-level query languages
2. lack of standardized interfaces,
3. huge previous investments in existing relational databases

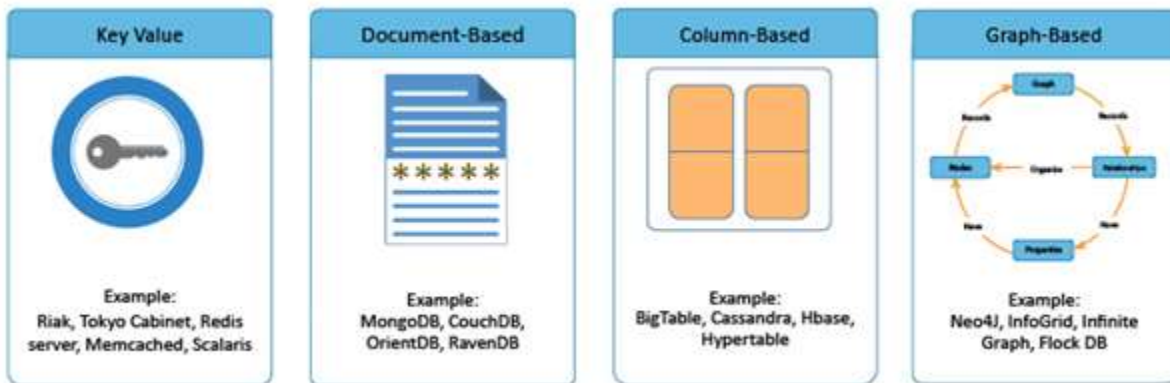
Most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes, so queries for data might not return updated data immediately or might result in reading data that is not accurate, this problem is known as stale reads.

Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss. Some NoSQL systems provide concepts such as write-ahead logging to avoid data loss.

## • 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 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 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).

It is not an ideal method if you are only looking to just update part of a value or query the database.

Example: Consider the data subset of IBM centers represented in the following table. Here the key is the name of the country, while the value is a list of addresses of 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"}
"US"	{"3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033"}

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 key.
- Put (key, value): associates the value with the key.
- Multi-get (key1, key 2..., key N): returns the list of values associated with the list of keys.
- Delete(key): removes the entry for the key from the data store.

#### Disadvantages:

- The model will not provide any kind of traditional database capabilities (such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously).
- If volume of the 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.

## 2. Document Store NoSQL Database

The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or 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.

```
{officeName:"IBM Noida",
{Street: "B-25, City:"Noida", State:"UP", Pincode:"201301"}
}
{officeName:"IBM imisoara",
```

```
{Boulevard:"Coriolan Brediceanu No. 10", Block:"B, Ist Floor", City: "Timisoara", Pincode: 300011"}  
}  
{officeName:"IBM Cluj",  
{Latitude:"40.748328", Longitude:"-73.985560"}  
}
```

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 that particular city.

### **3. Column Store NoSQL Database–**

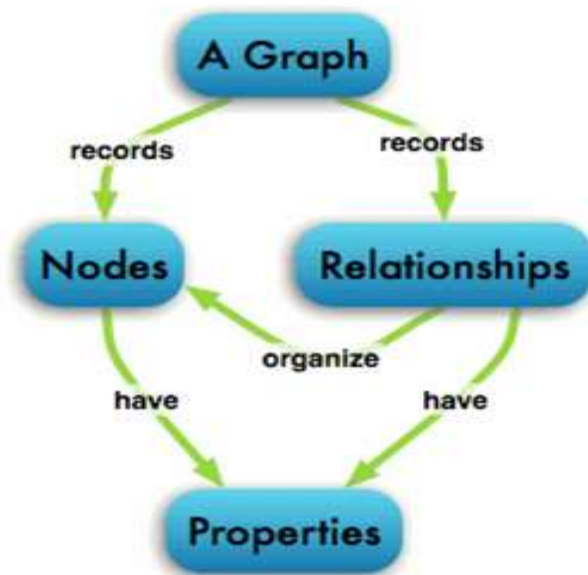
In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read 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. 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 articles is just one disk access, title of all the items can be obtained.

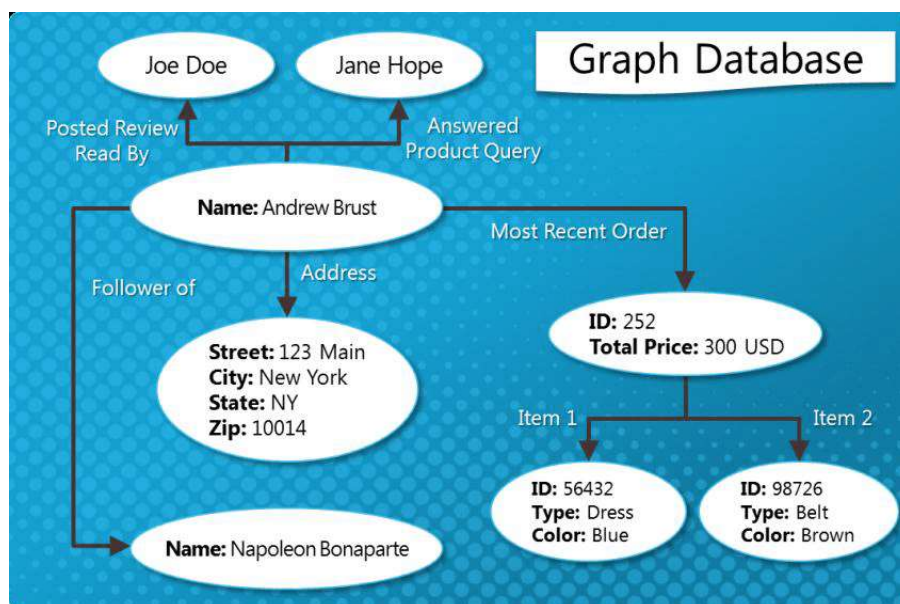
### **4. Graph Base NoSQL Database**

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.

Example: Amazon App, customer Andrew Brust who is a follower of Napoleon Bonaparte ordered a product with ID 252 and price \$300, which consists of 2 items and provides the delivery Address, this order was reviewed by Joe Doe and product query was answered by Jane Hope.



## • CAP Theorem

In a distributed system, the following three properties are important.

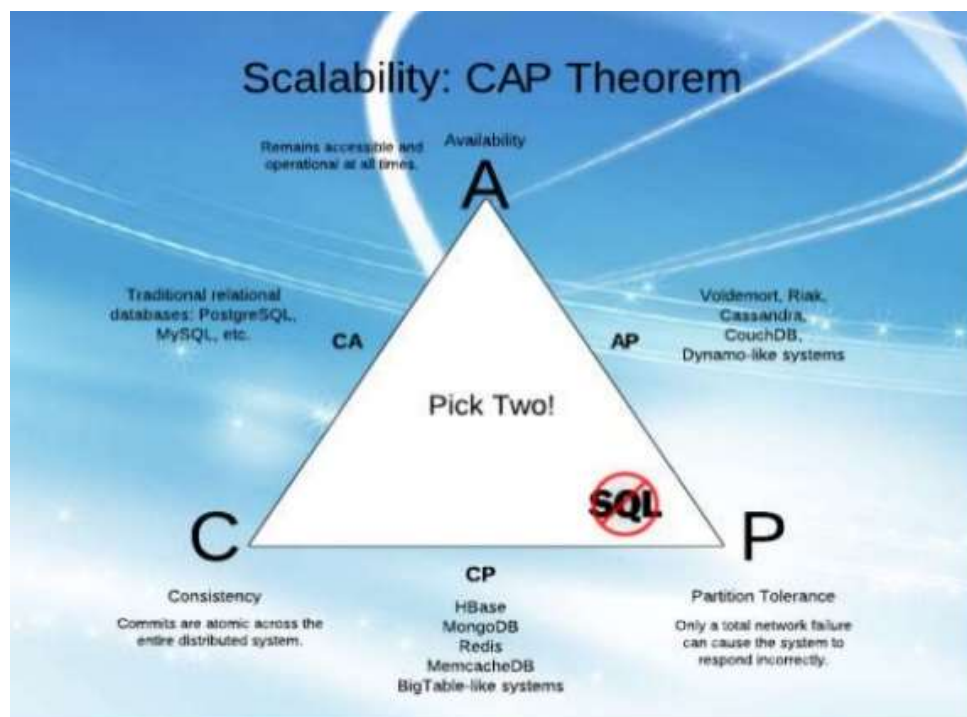
- Consistency: Each client must have consistent or the same view of the data.
- Availability: The data must be available to all clients for read and write operations.
- Partition toleration: System must work well across distributed networks.

The CAP theorem was proposed by Eric Brewer.

According to this theorem, in any distributed system, you can use only two of the three properties—consistency, availability, or partition tolerance simultaneously.

Many NoSQL databases provide options for a developer to choose to adjust the database as per requirement. For this, understanding the following requirements is important:

- How the data is consumed by the system?
- Whether the data is read or write heavy.
- If there is a need to query data with random parameters.
- If the system is capable of handling inconsistent data.



## Consistency

### 1. Consistency in CAP theorem

Consistency in CAP theorem refers to atomicity and isolation. Consistency means consistent read and write operations for the same sets of data so that concurrent operations see the same valid and consistent data state, without any stale data.

### 2. Consistency in ACID

Consistency in ACID means if the data does not satisfy predefined constraints, it is not persisted. Consistency in CAP theorem is different. In a single-machine database, consistency is achieved using the ACID semantics. However, in the case of NoSQL databases which are scaled out and distributed providing consistency gets complicated.

## Availability

According to the CAP theorem, availability means:

- The database system must be available to operate when required. This means that a system that is busy, uncommunicative, unresponsive, or inaccessible is not available.
- If a system is not available to serve a request at a time it is needed, it is unavailable.

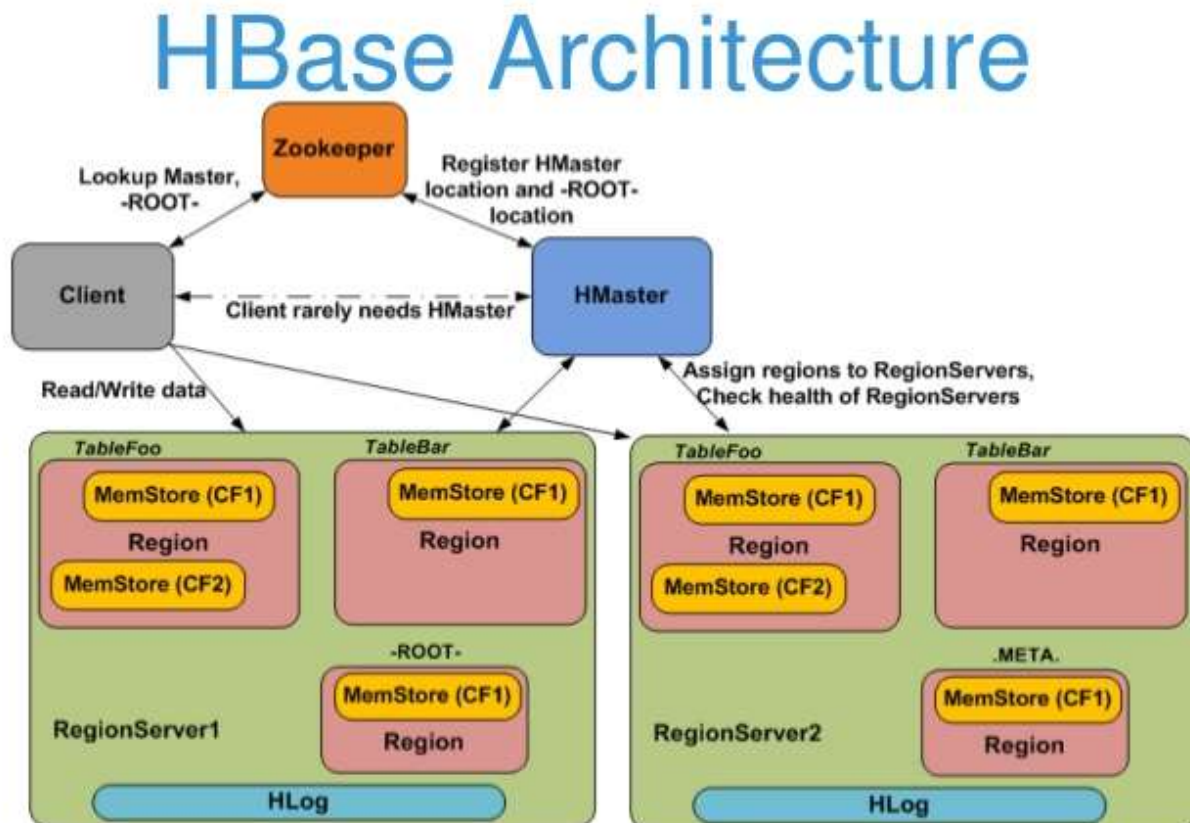
## Partition Tolerance

Partition tolerance or fault-tolerance is the third element of the CAP theorem. Partition tolerance measures the ability of a system to continue its service when some of its clusters become unavailable.



## • HBase Architecture

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.



Components of Apache HBase Architecture  
HBase architecture has 3 important components-

1. HMaster
2. Region Server
3. ZooKeeper

## **Hmaster:**

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

## **Region servers**

- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Decide the size of the region by following the region size thresholds.

Region Server runs on HDFS DataNode and consists of the following components –

- *Block Cache* – This is the read cache. Most frequently reads the data 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*- It is the actual storage file that stores the rows as sorted key values on a disk.

## **Zookeeper**

- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

## ● HBase vs RDBMS

Given below is a table that explains the difference between RDBMS and NoSQL.

RDBMS	NoSQL
<ul style="list-style-type: none"><li>• Data is stored in a relational model, with rows and columns.</li><li>• Row contains information about an item while columns contain specific information, such as 'Model', 'Date of Manufacture', 'Color'.</li><li>• Follows fixed schema. Meaning, the columns are defined and locked before data entry. In addition, each row contains data for each column.</li><li>• Supports vertical scaling. Scaling an RDBMS across multiple servers is a challenging and time-consuming process.</li><li>• Atomicity, Consistency, Isolation &amp; Durability(ACID) Compliant</li></ul>	<ul style="list-style-type: none"><li>• Data is stored in host of different databases, with different data storage models.</li><li>• Follows dynamic schemas. Meaning. You can add columns anytime.</li><li>• Supports horizontal scaling. You can scale across multiple servers. Multiple servers are cheap commodity hardware or cloud instances, which make scaling cost-effective compared to vertical scaling.</li><li>• Not ACID Compliant.</li></ul>

## Task 2: Execute blog present in below link

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

This blog, demonstrates how a small sample dataset data inside HDFS is loaded into HBase.

### Step 1:


- Check if all the nodes, Hregion server, Hmaster, HQuorumpeer, history server are active using the jps command
- Create a directory "hbase1" in HDFS using the command "Hadoop fs -mkdir /hbase1"
- Create a tsv file in local system under hbase directory named "bulk\_data.tsv" and move this tsv file from local filesystem to HDFS, using the command "Hadoop fs -put bulk\_data.tsv /hbase1/"

```

[acadgild@localhost ~]$ jps
2928 SecondaryNameNode
4162 HRegionServer
3155 ResourceManager
6117 Jps
2694 NameNode
2790 DataNode
3257 NodeManager
4284 Main
3660 JobHistoryServer
3966 HQuorumPeer
4062 HMaster
[acadgild@localhost ~]$ hadoop fs -ls /
18/04/10 18:48:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 6 items
drwxr-xr-x - acadgild supergroup      0 2018-03-26 19:38 /hadoopdata
drwxr-xr-x - acadgild supergroup      0 2018-04-10 18:41 /hbase
drwxr-xr-x - acadgild supergroup      0 2018-04-03 18:52 /home
drwxr-xr-x - acadgild supergroup      0 2018-02-02 12:49 /sqoopout111
drwxrwx--- - acadgild supergroup      0 2018-03-29 12:30 /tmp
drwxr-xr-x - acadgild supergroup      0 2018-04-08 15:58 /user
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ hadoop fs -mkdir /hbase1
18/04/10 18:48:59 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[acadgild@localhost ~]$ cd hbase
[acadgild@localhost hbase]$ hadoop fs -put bulk_data.tsv /hbase1/
18/04/10 18:49:50 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$

```

This the screen shot of bulk\_data.tsv file in local filesystem

 bulk\_data - Notepad

File	Edit	Format	View	Help
1		Amit	4	
2		Girija	3	
3		Jatin	5	
4		Swati	3	

This is the screen shot of the TSV file in HDFS:

```
[acadgild@localhost hbase]$ hadoop fs -cat /hbase1/bulk_data.tsv
18/04/18 18:53:28 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
1   Amit   4
2   Girija 3
3   Jatin  5
4   Swati  3[acadgild@localhost hbase]$
```

Contents of TSV file in HDFS

## Step 2:

Create table called “bulktable” with two column families “cf1 & cf2” in Hbase

```
hbase(main):007:0> create 'bulktable', 'cf1','cf2'
0 row(s) in 1.2570 seconds

=> Hbase::Table - bulktable
```

Created the table with two column families before importing data from HDFS

## Step 3:

We have to import the data present in HDFS to Hbase using the command:

**hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -  
Dimporttsv.columns=HBASE\_ROW\_KEY,cf1:name,cf2:exp bulktable  
/hbase1/bulk\_data.tsv**

This command will run a map reduce program to move the data from HDFS to “bulktable” in Hbase.



```

[acadgild@localhost hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE ROW KEY,cf1:name,cf2:exp bulkta
ble /hbase1/bulk data.tsv
2018-04-10 18:55:23,611 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-jav
a 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/StaticLoggerBind
er.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]
2018-04-10 18:55:24,878 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to ZooKeeper
ensemble=localhost:2181
2018-04-10 18:55:24,904 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09
GMT
2018-04-10 18:55:24,904 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-04-10 18:55:24,904 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-04-10 18:55:24,904 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-04-10 18:55:24,904 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-04-10 18:55:24,904 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hbase-1.2.6/c
onf:/usr/java/jdk1.8.0_151/lib/tools.jar:/home/acadgild/install/hbase/hbase-1.2.6:/home/acadgild/install/hbase/hbase-1.2.6/lib/activati
on-1.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/aopalliance-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-118n-
2.0.0-M15.jar:/home/acadgild/install/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/hbas
e-1.2.6/lib/asm-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/avro-1.7.4.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/common
s-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/lib/commons-compress-1.4.1.jar:/home/acadgild/install/hba
se/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/i
ninstall/hbase/hbase-1.2.6/lib/commons-httpclient-3.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commons-io-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/acadgild/
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/insta
ll/hbase/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/hbase-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/h
2018-04-10 18:55:24,911 INFO [main] zookeeper.ZooKeeper: Client environment:java.io.tmpdir=/tmp
2018-04-10 18:55:24,911 INFO [main] zookeeper.ZooKeeper: Client environment:java.compiler=<NA>
2018-04-10 18:55:24,911 INFO [main] zookeeper.ZooKeeper: Client environment:os.name=linux
2018-04-10 18:55:24,911 INFO [main] zookeeper.ZooKeeper: Client environment:os.arch=amd64
2018-04-10 18:55:24,911 INFO [main] zookeeper.ZooKeeper: Client environment:os.version=2.6.32-696.23.1.el6.x86_64
2018-04-10 18:55:24,912 INFO [main] zookeeper.ZooKeeper: Client environment:user.name=acadgild
2018-04-10 18:55:24,912 INFO [main] zookeeper.ZooKeeper: Client environment:user.home=/home/acadgild
2018-04-10 18:55:24,912 INFO [main] zookeeper.ZooKeeper: Client environment:user.dir=/home/acadgild/hbase
2018-04-10 18:55:24,914 INFO [main] zookeeper.ZooKeeper: Initiating client connection, connectString=localhost:2181 sessionTimeout=900
00 watcher=hconnection-0x6025e1b6x0, quorum=localhost:2181, baseZNode=/hbase
2018-04-10 18:55:25,003 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)
2018-04-10 18:55:25,044 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Socket connection established to localhost/127.0.
0.1:2181, initiating session
2018-04-10 18:55:25,075 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Session establishment complete on server localhos
t/127.0.0.1:2181, sessionId = 0x162afae38c1000a, negotiated timeout = 90000
2018-04-10 18:55:27,290 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes.per.checksu
m
2018-04-10 18:55:27,456 INFO [main] client.ConnectionManager$HConnectionImplementation: Closing zookeeper sessionId=0x162afae38c1000a
2018-04-10 18:55:27,462 INFO [main-EventThread] zookeeper.ClientCnxn: EventThread shut down
2018-04-10 18:55:27,462 INFO [main] zookeeper.ZooKeeper: Session: 0x162afae38c1000a closed
2018-04-10 18:55:27,697 INFO [main] client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
2018-04-10 18:55:28,180 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes.per.checksu
m
2018-04-10 18:55:31,350 INFO [main] input.FileInputFormat: Total input paths to process : 1
2018-04-10 18:55:31,494 INFO [main] mapreduce.JobSubmitter: number of splits:1
2018-04-10 18:55:31,521 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes.per.checksu
m
2018-04-10 18:55:31,834 INFO [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1523365705607_0002
2018-04-10 18:55:32,315 INFO [main] impl.YarnClientImpl: Submitted application application_1523365705607_0002
2018-04-10 18:55:32,384 INFO [main] mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1523365705607_000
2/
2018-04-10 18:55:32,386 INFO [main] mapreduce.Job: Running job: job_1523365705607_0002
2018-04-10 18:55:47,885 INFO [main] mapreduce.Job: Job job_1523365705607_0002 running in uber mode : false
2018-04-10 18:55:47,892 INFO [main] mapreduce.Job: map 0% reduce 0%
2018-04-10 18:56:01,452 INFO [main] mapreduce.Job: map 100% reduce 0%
2018-04-10 18:56:01,476 INFO [main] mapreduce.Job: Job job_1523365705607_0002 completed successfully
2018-04-10 18:56:01,723 INFO [main] mapreduce.Job: Counters: 31

```

```

FILE: Number of bytes read=0
FILE: Number of bytes written=139464
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=149
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)=10204
  Total time spent by all reduces in occupied slots (ms)=0
  Total time spent by all map tasks (ms)=10204
  Total vcore-seconds taken by all map tasks=10204
  Total megabyte-seconds taken by all map tasks=10448096
Map-Reduce Framework
  Map input records=4
  Map output records=4
  Input split bytes=107
  Spilled Records=0
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=179
  CPU time spent (ms)=2770
  Physical memory (bytes) snapshot=116248576
  Virtual memory (bytes) snapshot=2067746816
  Total committed heap usage (bytes)=32571392
ImportTsv
  Bad Lines=0
  File Input Format Counters
    Bytes Read=42
  File Output Format Counters
    Bytes Written=0
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$

```

Now we can notice that the data was successfully transferred. We will check if the data is transferred in Hbase, using the Scan command in Hbase.

```

hbase(main):008:0> scan 'bulktable'
ROW          COLUMN+CELL
1            column=cf1:name, timestamp=1523366723471, value=Amit
1            column=cf2:exp, timestamp=1523366723471, value=4
2            column=cf1:name, timestamp=1523366723471, value=Girija
2            column=cf2:exp, timestamp=1523366723471, value=3
3            column=cf1:name, timestamp=1523366723471, value=Jatin
3            column=cf2:exp, timestamp=1523366723471, value=5
4            column=cf1:name, timestamp=1523366723471, value=Swati
4            column=cf2:exp, timestamp=1523366723471, value=3
4 row(s) in 0.1030 seconds

hbase(main):009:0>

```

Data is transferred from HDFS to Hbase

We can see that all the data from “bulk\_data.tsv” files are successfully transferred to “bulktable” in Hbase.