Session 11:
ADVANCE HBASE

Assignment 1

- By Prachi Mohite

__Task 1__

Explain the below concepts with an example in brief.

1.1 Nosql Databases

Answer:

NoSQL is 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 databases are especially useful for working with large sets of distributed data.

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.

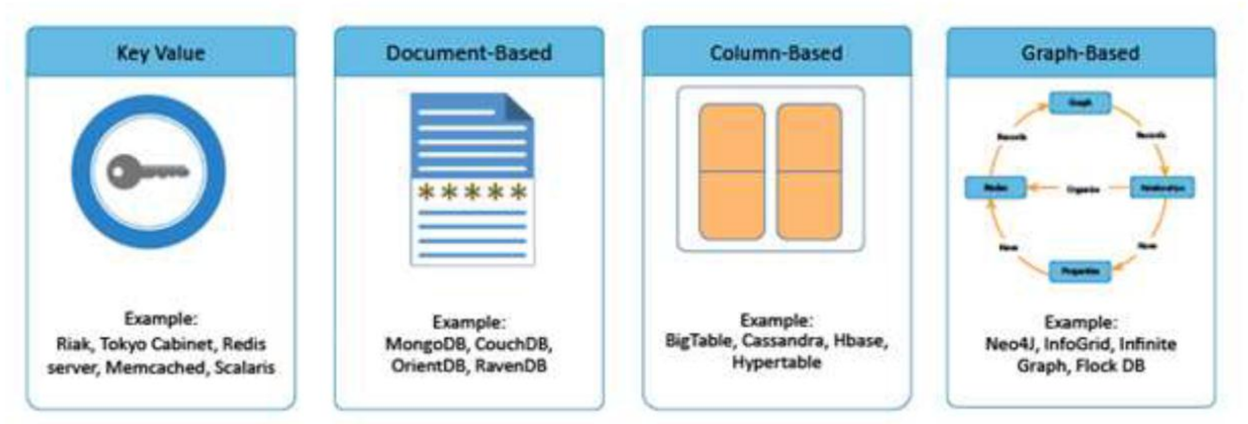
Most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes, so queries for data might not return updated data immediately or might result in reading data that is not accurate, this problem is known as stale reads.

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.

NoSQL Database Types

There are 4 basic 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. It stores documents made up of tagged elements. {Example-CouchDB}
- **Graph stores** are used to store information about networks of data, such as social connections. A network database that uses edges and nodes to represent and store data. {Example- Neo4J}
- **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. It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
- **Wide-column** stores such as HBase are optimized for queries over large datasets, and store columns of data together, instead of rows. Each storage block contains data from only one column, {Example- HBase, Cassandra}


Key Value
Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris
Document-Based
Example:
MongoDB, CouchDB, OrientDB, RavenDB
Column-Based
Example:
BigTable, Cassandra, Hbase, Hypertable
Graph-Based
Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

1.2 Types of Nosql Databases

1. Document Store No SQL Database

The central concept of a document store is the notion of a "document". While each documentoriented
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.

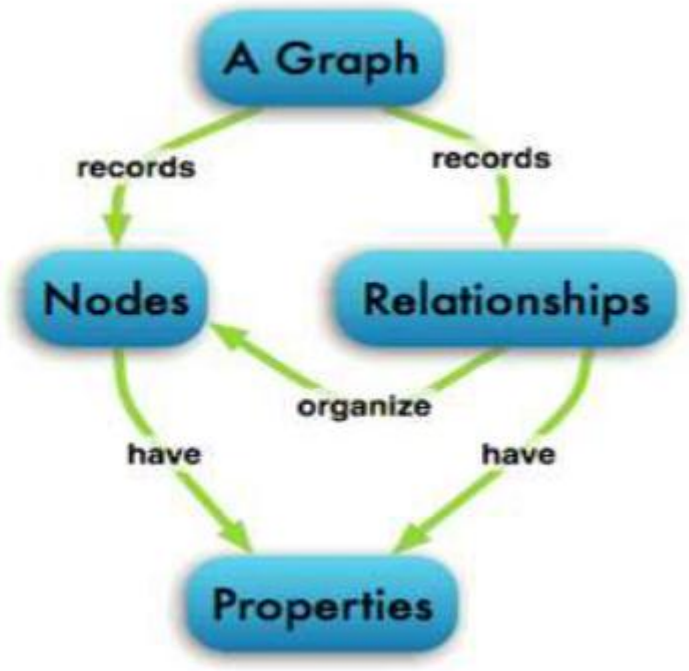
{officeName:”IBM Noida”,

```
{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.

1 Graph Store 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.

2 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

- 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.

3 Wide-Column store 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.

1.3CAP Theorem

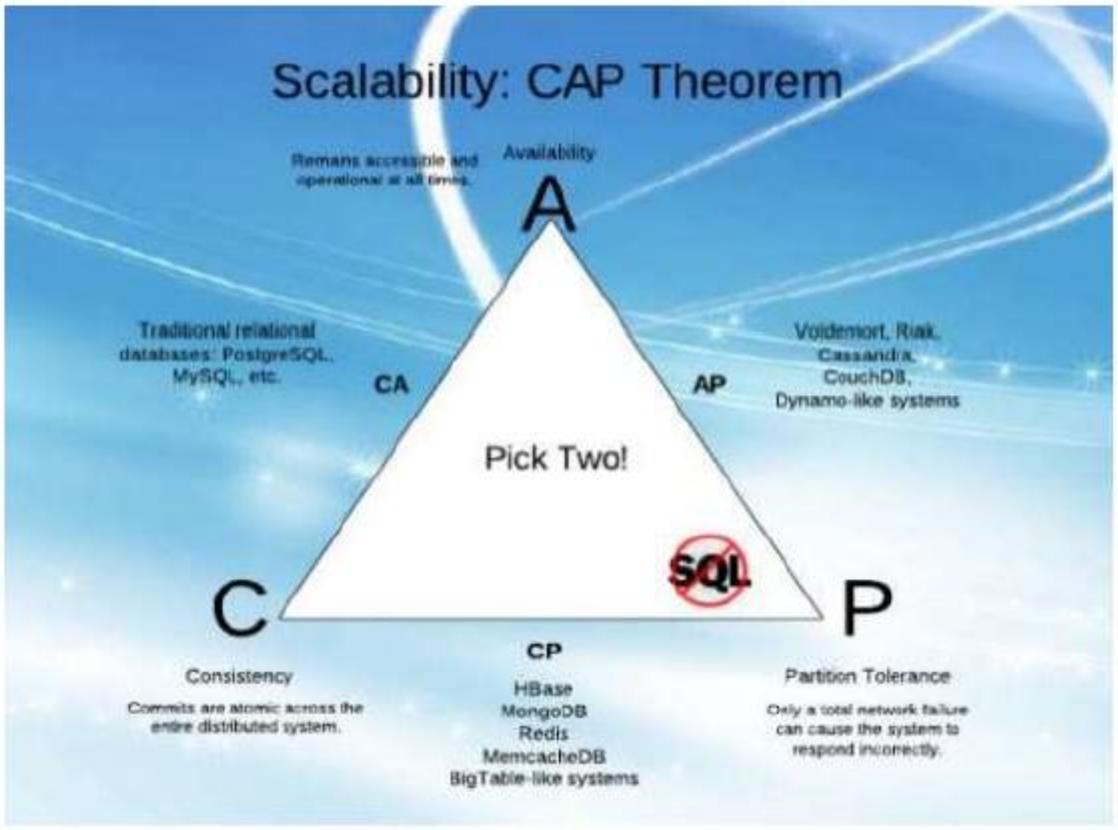
In a distributed system, the following three properties are important.

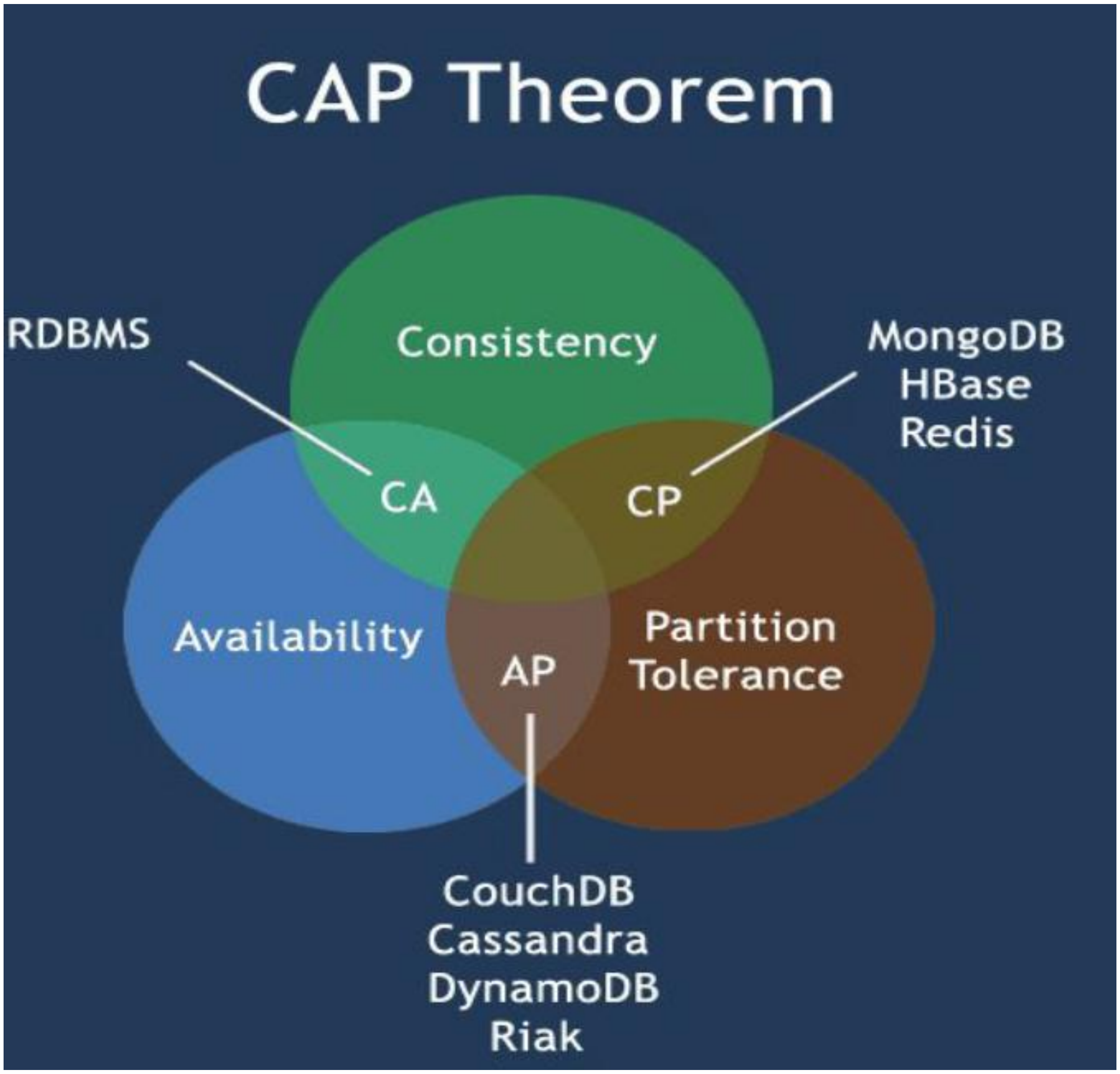
- **Consistency** - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation, all clients see the same data.
- **Availability** - This means that the system is always on (service guarantee availability), no downtime.
- **Partition Tolerance** - This means that the system continues to function even if the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

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.


Scalability: CAP Theorem
Remains accessible and operational at all times.
Availability
A
Traditional relational databases: PostgreSQL, MySQL, etc.
CA
AP
Voldemort, Riak, Cassandra, CouchDB, Dynamo-like systems
Pick Two!
SQL
C
P
CP
Consistency
Commits are atomic across the entire distributed system.
HBase
MongoDB
Redis
MemcacheDB
BigTable-like systems
Partition Tolerance
Only a total network failure can cause the system to respond incorrectly.



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:

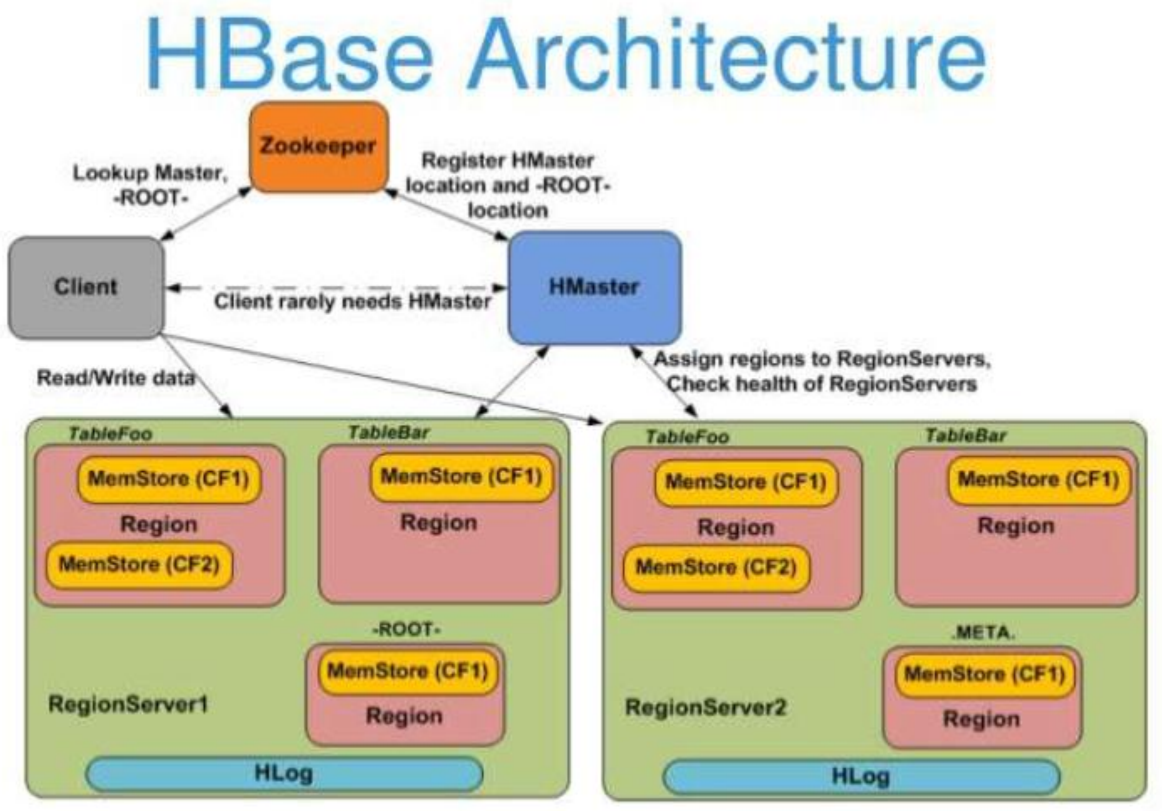
- The database system must be available to operate when required. This means that a system that is busy, uncommunicative, unresponsive, or inaccessible is not available.
- If a system is not available to serve a request at a time it is needed, it is unavailable.

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.

1.4 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 (Master Server)
2. Region Servers
3. ZooKeeper

Master Server (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.

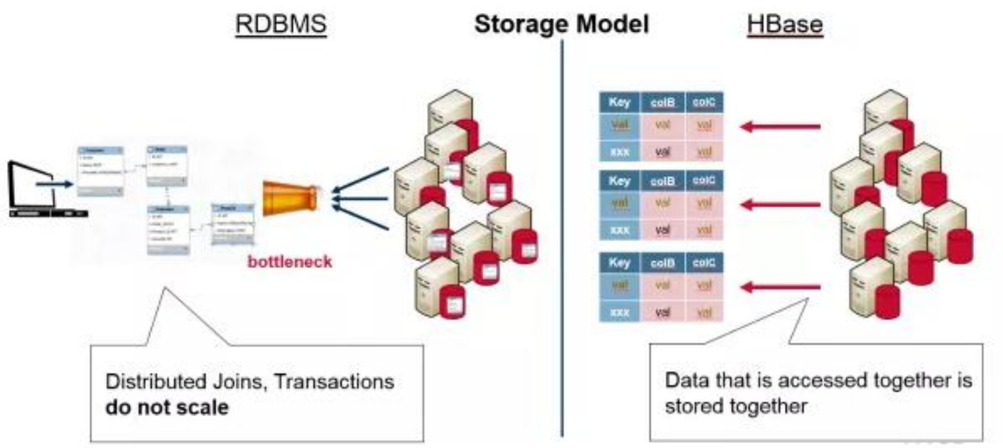
Zookeeper

- 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.

1.5 HBase vs RDBMS

HBase	RDBMS
1. Column-oriented	1. Row-oriented(mostly)
2. Flexible schema, add columns on the	2. Fixed schema

Fly	
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 war.	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.

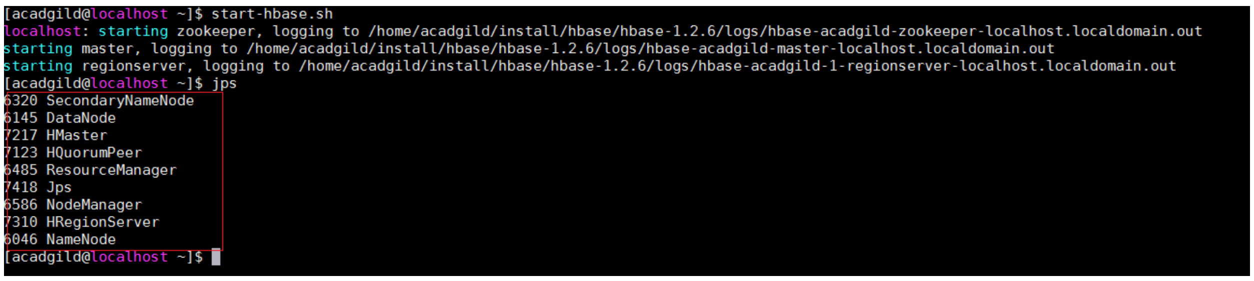

RDBMS
Storage Model
HBase
bottleneck
Key colB colC
val val val
xxx val val
Distributed Joins, Transactions do not scale
Data that is accessed together is stored together

Task 2

Execute blog present in below link

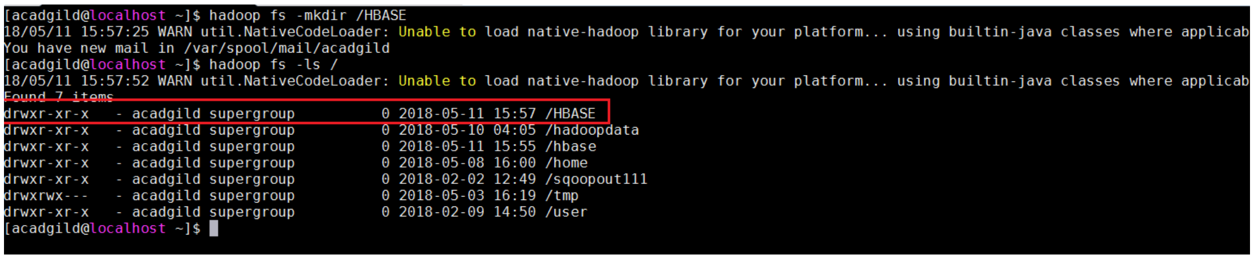
https://acadgild.com/blog/importtsv-data-from-hdfs-into-hbase/

- start all the Hadoop and HBase daemons



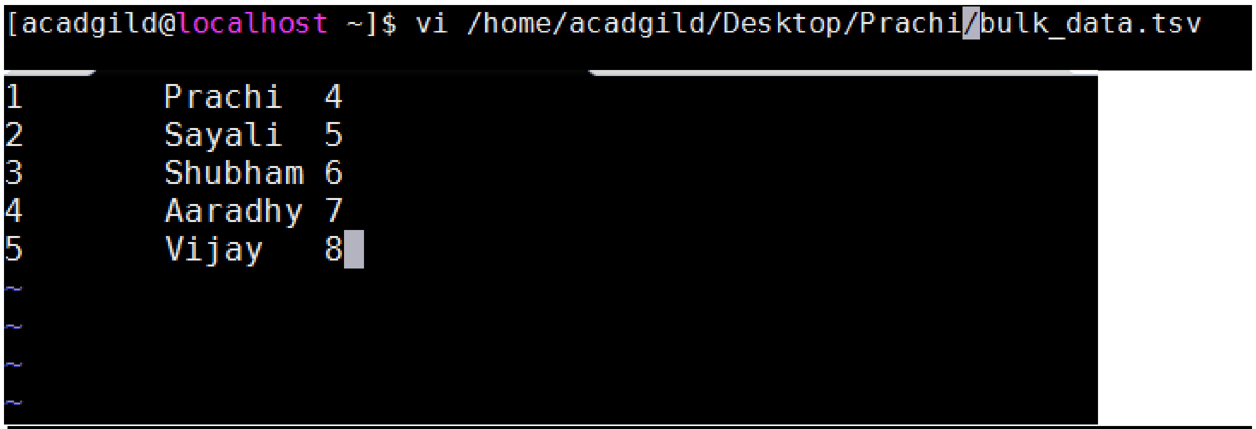
```
[acadgild@localhost ~]$ start-hbase.sh
localhost: starting zookeeper, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-acadgild-zookeeper-localhost.localdomain.out
starting master, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-acadgild-master-localhost.localdomain.out
starting regionserver, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-acadgild-1-regionserver-localhost.localdomain.out
[acadgild@localhost ~]$ jps
6320 SecondaryNameNode
6145 DataNode
7217 HMaster
7123 HQuorumPeer
6485 ResourceManager
7418 Jps
6586 NodeManager
7310 HRegionServer
6046 NameNode
[acadgild@localhost ~]$
```

- As we need to load data from HDFS create a directory in HDFS to name given as HBASE
    - Command used to achieve the same is mkdir



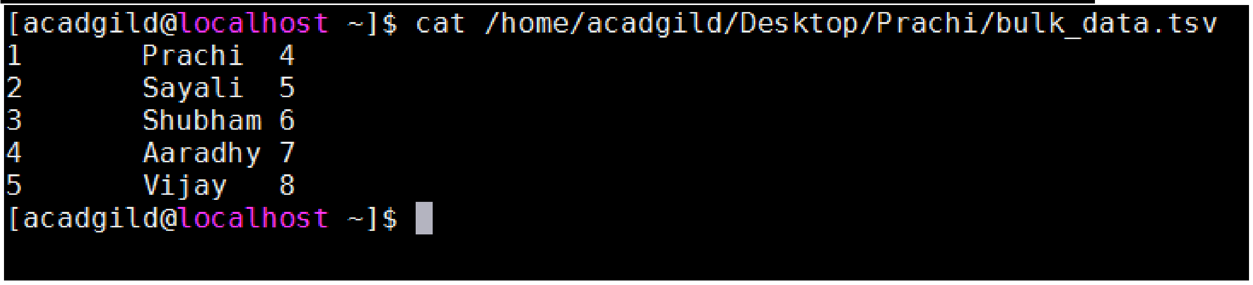
```
[acadgild@localhost ~]$ hadoop fs -mkdir /HBASE
18/05/11 15:57:25 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicab
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ hadoop fs -ls /
18/05/11 15:57:52 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicab
Found 7 items
drwxr-xr-x   - acadgild supergroup          0 2018-05-11 15:57 /HBASE
drwxr-xr-x   - acadgild supergroup          0 2018-05-10 04:05 /hadoopdata
drwxr-xr-x   - acadgild supergroup          0 2018-05-11 15:55 /hbase
drwxr-xr-x   - acadgild supergroup          0 2018-05-08 16:00 /home
drwxr-xr-x   - acadgild supergroup          0 2018-02-02 12:49 /sqoopout111
drwxrwx---   - acadgild supergroup          0 2018-05-03 16:19 /tmp
drwxr-xr-x   - acadgild supergroup          0 2018-02-09 14:50 /user
[acadgild@localhost ~]$
```

- Create a CSV file and move that file to HBASE directory in HDFS
    - Command used to create a file is vi



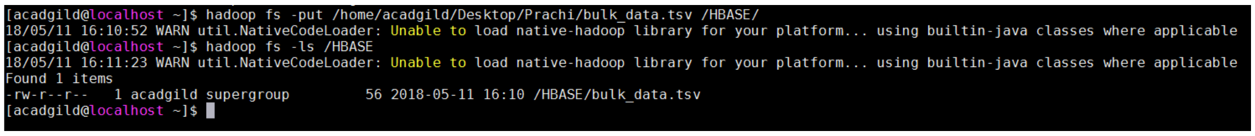
```
[acadgild@localhost ~]$ vi /home/acadgild/Desktop/Prachi/bulk_data.tsv
```

```
1       Prachi  4
2       Sayali  5
3       Shubham 6
4       Aaradhy 7
5       Vijay   8
~
~
~
~
```



```
[acadgild@localhost ~]$ cat /home/acadgild/Desktop/Prachi/bulk_data.tsv
1       Prachi  4
2       Sayali  5
3       Shubham 6
4       Aaradhy 7
5       Vijay   8
[acadgild@localhost ~]$
```

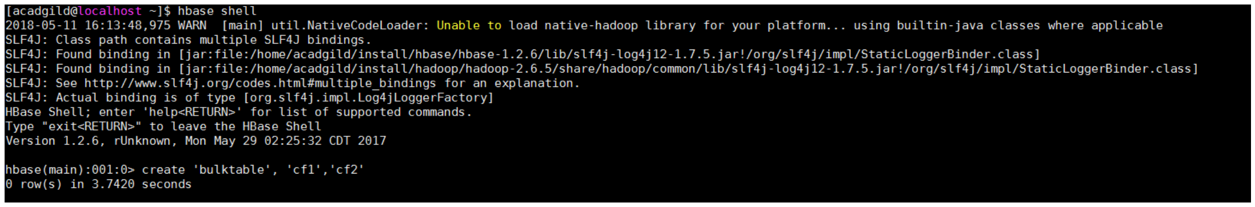
- Command used to move that file put command



```
[acadgild@localhost ~]$ hadoop fs -put /home/acadgild/Desktop/Prachi/bulk_data.tsv /HBASE/
18/05/11 16:10:52 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[acadgild@localhost ~]$ hadoop fs -ls /HBASE
18/05/11 16:11:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 acadgild supergroup         56 2018-05-11 16:10 /HBASE/bulk_data.tsv
[acadgild@localhost ~]$
```

- Launch hbase shell

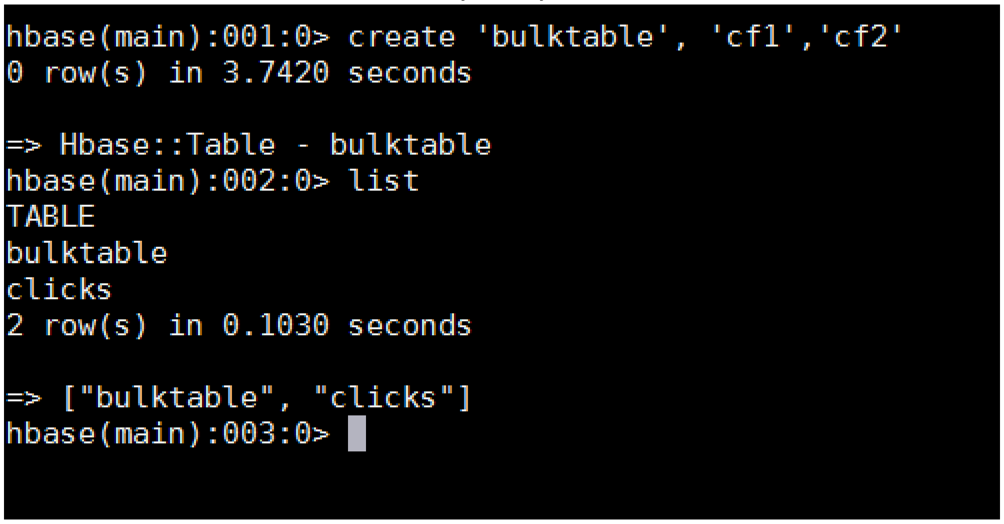
-



```
[acadgild@localhost ~]$ hbase shell
2018-05-11 16:13:48,975 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/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; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> create 'bulktable', 'cf1','cf2'
0 row(s) in 3.7420 seconds
```

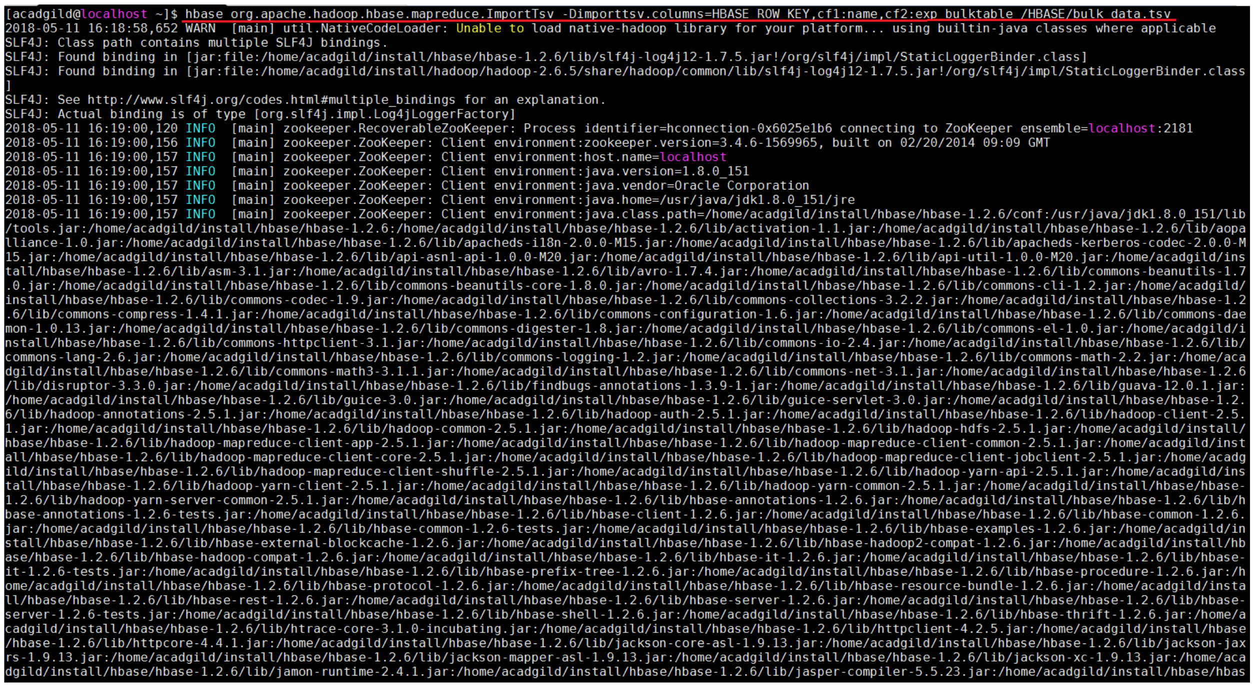
- Create a table with two column family

  - **Create 'bulktable', 'cf1', 'cf2'**

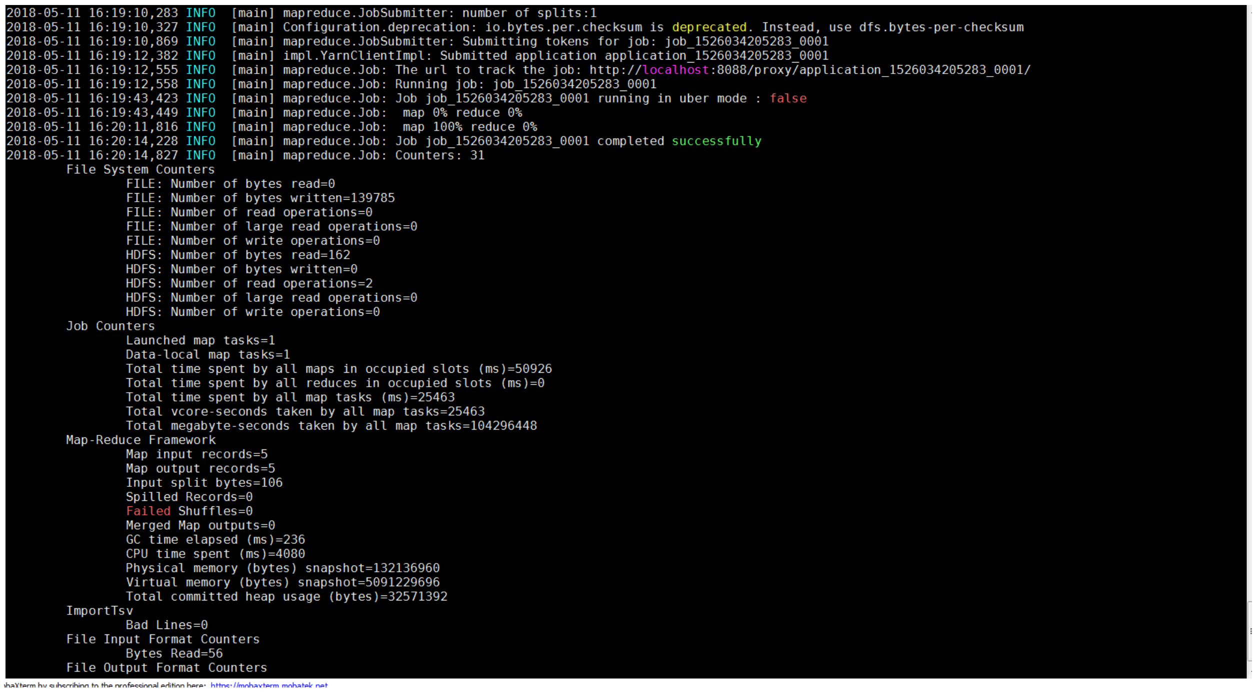


```
hbase(main):001:0> create 'bulktable', 'cf1','cf2'
0 row(s) in 3.7420 seconds

=> Hbase::Table - bulktable
hbase(main):002:0> list
TABLE
bulktable
clicks
2 row(s) in 0.1030 seconds

=> ["bulktable", "clicks"]
hbase(main):003:0>
```

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



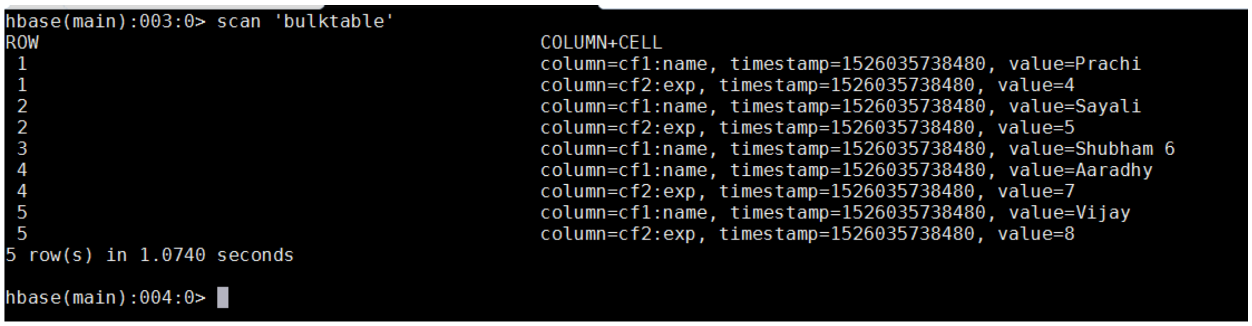
```
[acadgild@localhost ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /HBASE/bulk_data.tsv
2018-05-11 16:18:58,652 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes 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/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]
2018-05-11 16:19:00,120 INFO  [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to ZooKeeper ensemble=localhost:2181
2018-05-11 16:19:00,156 INFO  [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT
2018-05-11 16:19:00,157 INFO  [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-05-11 16:19:00,157 INFO  [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-05-11 16:19:00,157 INFO  [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-05-11 16:19:00,157 INFO  [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-05-11 16:19:00,157 INFO  [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hbase-1.2.6/conf:/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/activation-1.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/aopa
lliance-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-i18n-2.0.0-M15.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-kerberos-codec-2.0.0-M
15.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/ins
tall/hbase/hbase-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/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/lib/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-dae
mon-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
nstall/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/aca
dgild/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/install/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/hbase/hbase-1.2.
6/lib/hadoop-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-client-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-mapreduce-client-common-2.5.1.jar:/home/acadgild/inst
all/hbase/hbase-1.2.6/lib/hadoop-mapreduce-client-core-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-mapreduce-client-jobclient-2.5.1.jar:/home/acadg
ild/install/hbase/hbase-1.2.6/lib/hadoop-mapreduce-client-shuffle-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-yarn-api-2.5.1.jar:/home/acadgild/ins
tall/hbase/hbase-1.2.6/lib/hadoop-yarn-client-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hadoop-yarn-common-2.5.1.jar:/home/acadgild/install/hbase/hbase-
1.2.6/lib/hadoop-yarn-server-common-2.5.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-annotations-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/h
base-annotations-1.2.6-tests.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-client-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-common-1.2.6.
jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-common-1.2.6-tests.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-examples-1.2.6.jar:/home/acadgild/in
stall/hbase/hbase-1.2.6/lib/hbase-external-blockcache-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-hadoop2-compat-1.2.6.jar:/home/acadgild/install/hb
ase/hbase-1.2.6/lib/hbase-hadoop-compat-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-it-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-
it-1.2.6-tests.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-prefix-tree-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-procedure-1.2.6.jar:/h
ome/acadgild/install/hbase/hbase-1.2.6/lib/hbase-protocol-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-resource-bundle-1.2.6.jar:/home/acadgild/insta
ll/hbase/hbase-1.2.6/lib/hbase-rest-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-server-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-
server-1.2.6-tests.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-shell-1.2.6.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/hbase-thrift-1.2.6.jar:/home/a
cadgild/install/hbase/hbase-1.2.6/lib/htrace-core-3.1.0-incubating.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/httpclient-4.2.5.jar:/home/acadgild/install/hbase
/hbase-1.2.6/lib/httpcore-4.4.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/jackson-core-asl-1.9.13.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/jackson-jax
rs-1.9.13.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/jackson-mapper-asl-1.9.13.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/jackson-xc-1.9.13.jar:/home/aca
dgild/install/hbase/hbase-1.2.6/lib/jamon-runtime-2.4.1.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/jasper-compiler-5.5.23.jar:/home/acadgild/install/hbase/hbas
```



```
2018-05-11 16:19:10,283 INFO  [main] mapreduce.JobSubmitter: number of splits:1
2018-05-11 16:19:10,327 INFO  [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-05-11 16:19:10,869 INFO  [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1526034205283_0001
2018-05-11 16:19:12,382 INFO  [main] impl.YarnClientImpl: Submitted application application_1526034205283_0001
2018-05-11 16:19:12,555 INFO  [main] mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1526034205283_0001/
2018-05-11 16:19:12,558 INFO  [main] mapreduce.Job: Running job: job_1526034205283_0001
2018-05-11 16:19:43,423 INFO  [main] mapreduce.Job: Job job_1526034205283_0001 running in uber mode : false
2018-05-11 16:19:43,449 INFO  [main] mapreduce.Job:  map 0% reduce 0%
2018-05-11 16:20:11,816 INFO  [main] mapreduce.Job:  map 100% reduce 0%
2018-05-11 16:20:14,228 INFO  [main] mapreduce.Job: Job job_1526034205283_0001 completed successfully
2018-05-11 16:20:14,827 INFO  [main] mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=139785
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=162
                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)=50926
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=25463
                Total vcore-seconds taken by all map tasks=25463
                Total megabyte-seconds taken by all map tasks=104296448
        Map-Reduce Framework
                Map input records=5
                Map output records=5
                Input split bytes=106
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=236
                CPU time spent (ms)=4080
                Physical memory (bytes) snapshot=132136960
                Virtual memory (bytes) snapshot=5091229696
                Total committed heap usage (bytes)=32571392
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=56
        File Output Format Counters
```

haXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

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):003:0> scan 'bulktable'
ROW                                              COLUMN+CELL
 1                                               column=cf1:name, timestamp=1526035738480, value=Prachi
 1                                               column=cf2:exp, timestamp=1526035738480, value=4
 2                                               column=cf1:name, timestamp=1526035738480, value=Sayali
 2                                               column=cf2:exp, timestamp=1526035738480, value=5
 3                                               column=cf1:name, timestamp=1526035738480, value=Shubham 6
 4                                               column=cf1:name, timestamp=1526035738480, value=Aaradhy
 4                                               column=cf2:exp, timestamp=1526035738480, value=7
 5                                               column=cf1:name, timestamp=1526035738480, value=Vijay
 5                                               column=cf2:exp, timestamp=1526035738480, value=8
5 row(s) in 1.0740 seconds

hbase(main):004:0> 
```

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