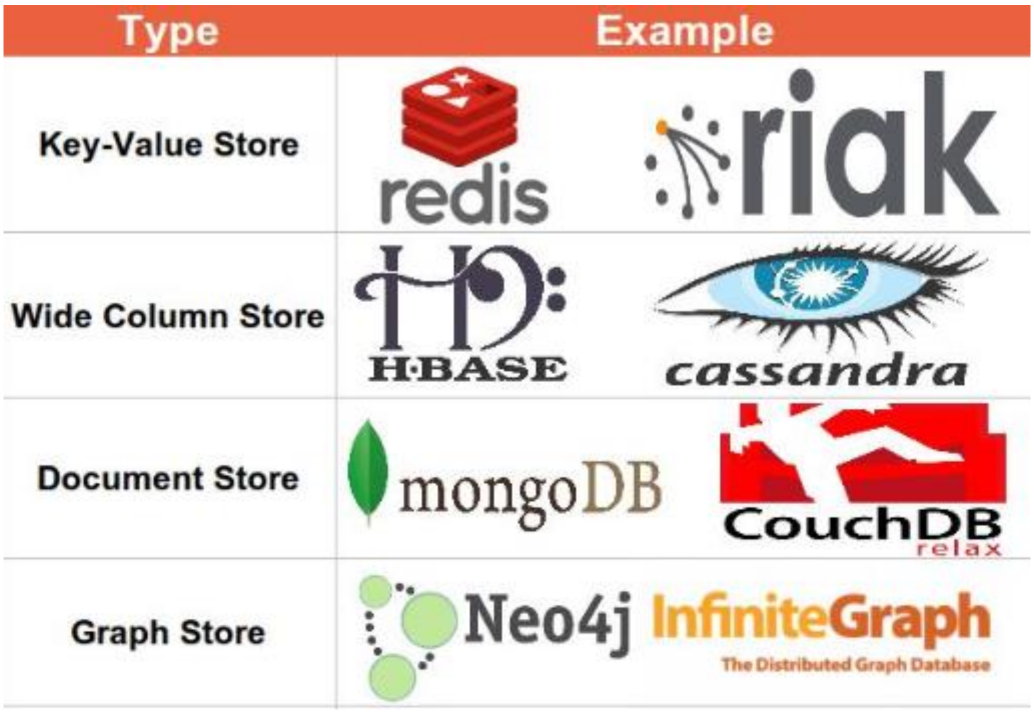
Task1

Explain the below concepts with an example in brief.

NoSQL is a class of database management systems (DBMS) that do not follow all of the rules of a relational DBMS and cannot use traditional SQL to query data.

NoSQL-based systems are typically used in very large databases, which are particularly prone to performance problems caused by the limitations of SQL and the relational model of databases.

NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images



Type	Example	
Key-Value Store	redis	riak
Wide Column Store	HBASE	cassandra
Document Store	mongoDB	CouchDB relax
Graph Store	Neo4j	InfiniteGraph The Distributed Graph Database

Types of Nosql Databases :

TYPES OF NOSQL DATABASES

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

Key-value data stores: Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.

Document stores: Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.

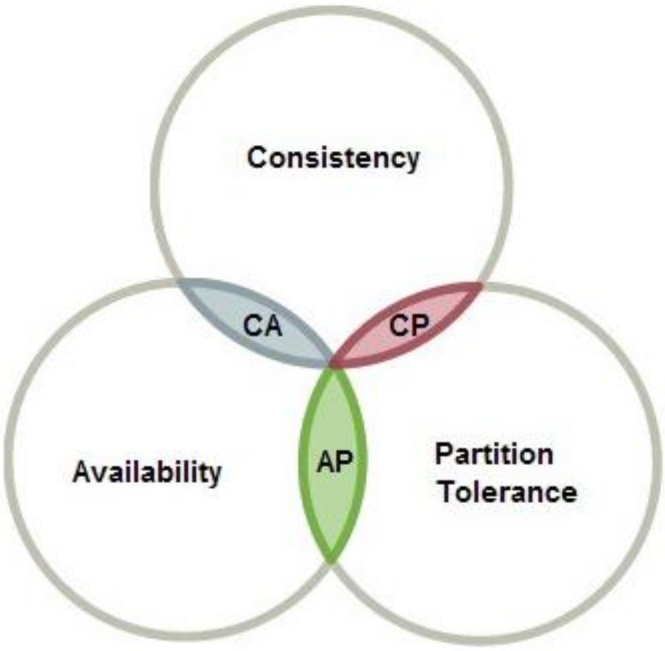
Wide-column stores: Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

Graph stores: A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

CAP Theorem:

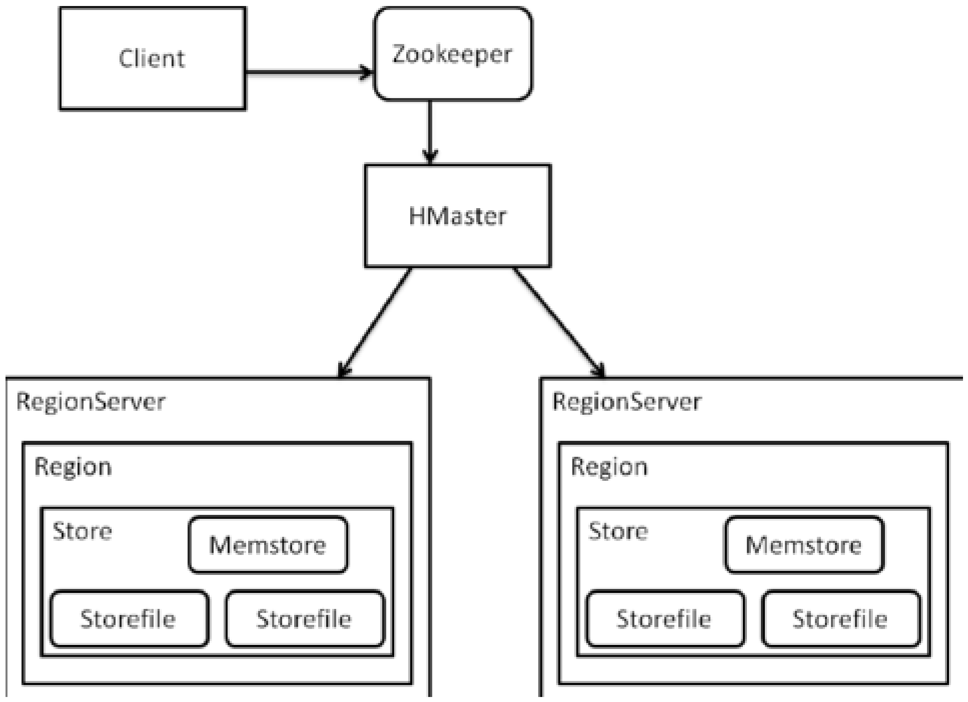
CAP Theorem is very important in the Big Data world, especially when we need to make trade off's between the three, based on our unique use case. The theorem states that **networked shared-data systems** can only guarantee/strongly support two of the following three properties:

- **Consistency** - A guarantee that every node in a distributed cluster returns the same, most recent, successful write. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.
- **Availability** - Every non-failing node returns a response for all read and write requests in a reasonable amount of time. The key word here is every. To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.
- **Partition Tolerant** - The system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.



HBase Architecture

HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.



MasterServer

The master server -

- 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.

Regions

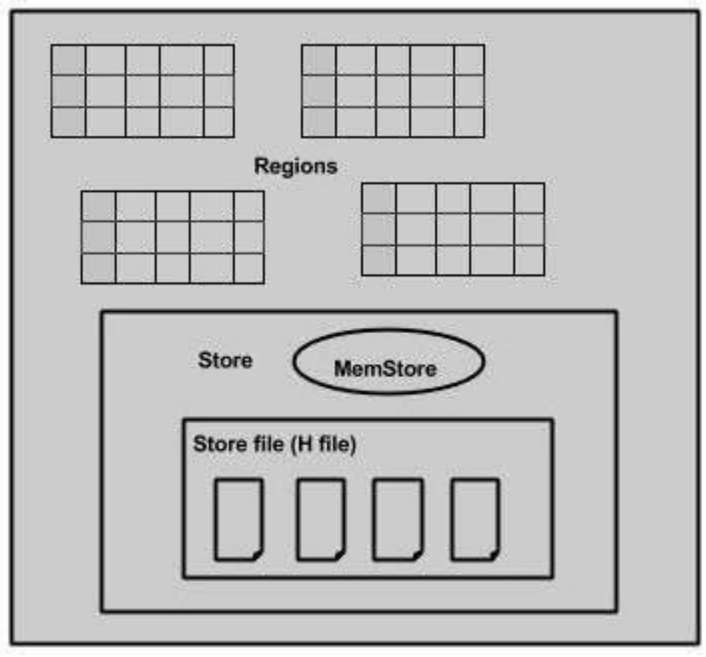
Regions are nothing but tables that are split up and spread across the region servers.

Region server

The region servers have regions that -

- 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.

When we take a deeper look into the region server, it contain regions and stores as shown below:



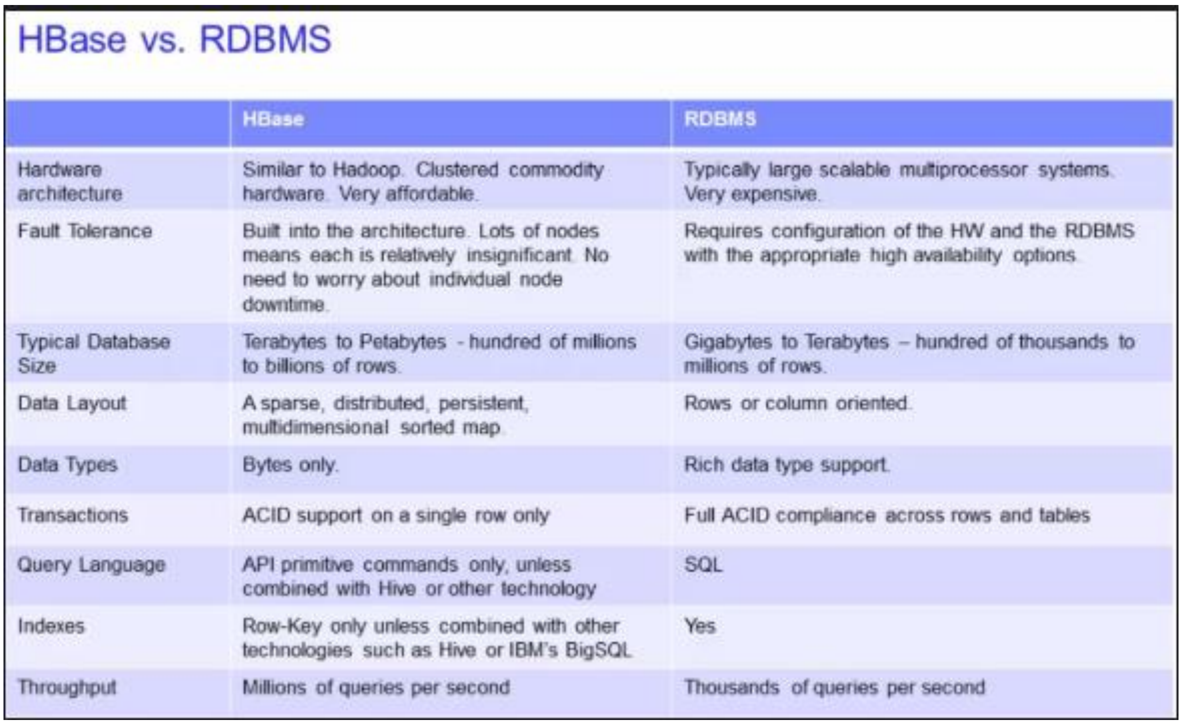
The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

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



HBase vs. RDBMS

	HBase	RDBMS
Hardware architecture	Similar to Hadoop. Clustered commodity hardware. Very affordable.	Typically large scalable multiprocessor systems. Very expensive.
Fault Tolerance	Built into the architecture. Lots of nodes means each is relatively insignificant. No need to worry about individual node downtime.	Requires configuration of the HW and the RDBMS with the appropriate high availability options.
Typical Database Size	Terabytes to Petabytes - hundred of millions to billions of rows.	Gigabytes to Terabytes – hundred of thousands to millions of rows.
Data Layout	A sparse, distributed, persistent, multidimensional sorted map.	Rows or column oriented.
Data Types	Bytes only.	Rich data type support.
Transactions	ACID support on a single row only	Full ACID compliance across rows and tables
Query Language	API primitive commands only, unless combined with Hive or other technology	SQL
Indexes	Row-Key only unless combined with other technologies such as Hive or IBM's BigSQL	Yes
Throughput	Millions of queries per second	Thousands of queries per second

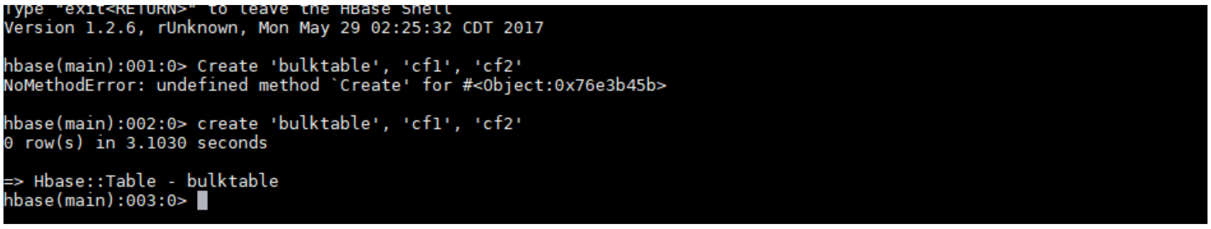
H Base	RDBMS
1. Column-oriented	1. Row-oriented(mostly)
2. Flexible schema, add columns on the Fly	2. Fixed schema
3. Good with sparse tables.	3. Not optimized for sparse tables.
4. No query language	4. SQL
5. Wide tables	5. Narrow tables
6. Joins using MR – not optimized	6. optimized for Joins(small, fast ones)
7. Tight – Integration with MR	7. Not really
8. De-normalize your data.	8. Normalize as you can
9. Horizontal scalability-just add hard 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.

Task2
Execute blog present in below link

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

Step1:

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

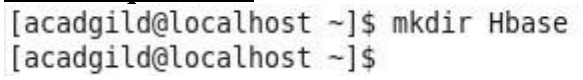


```
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> Create 'bulktable', 'cf1', 'cf2'
NoMethodError: undefined method `Create' for #<Object:0x76e3b45b>

hbase(main):002:0> create 'bulktable', 'cf1', 'cf2'
0 row(s) in 3.1030 seconds

=> Hbase::Table - bulktable
hbase(main):003:0>
```

Step2 :

Come out of HBase shell to the terminal and also make a directory for Hbase in the local drive; So,
since you have your own path you can use it.

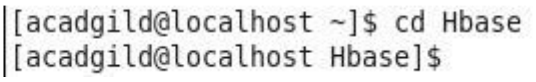
mkdir -p hbase



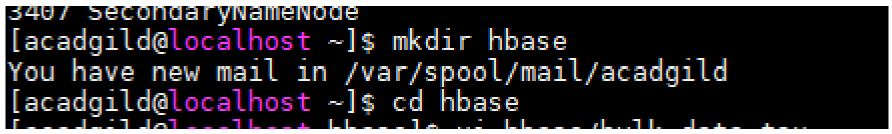
```
[acadgild@localhost ~]$ mkdir Hbase
[acadgild@localhost ~]$
```

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

cd hbase



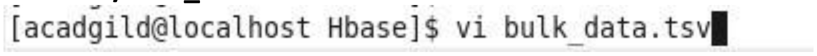
```
[acadgild@localhost ~]$ cd Hbase
[acadgild@localhost Hbase]$
```



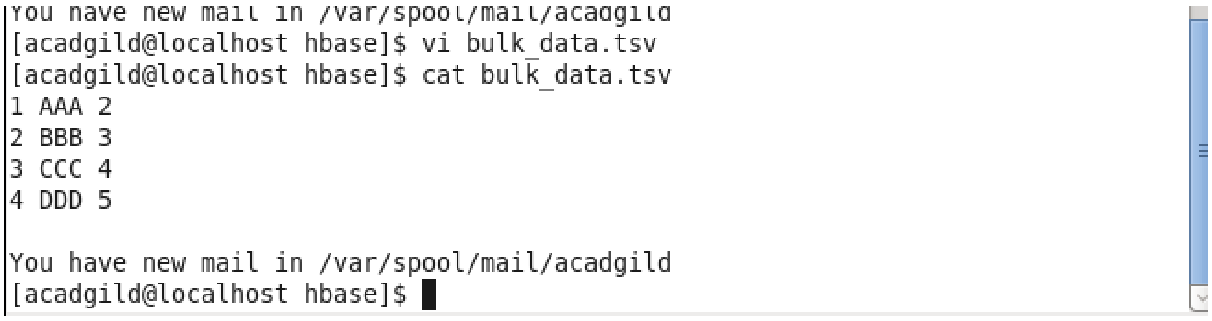
```
3407 SecondaryNameNode
[acadgild@localhost ~]$ mkdir hbase
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ cd hbase
```

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



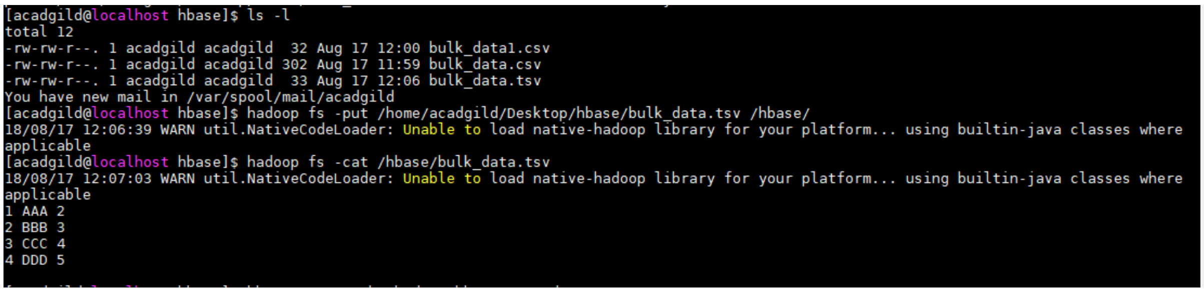
```
[acadgild@localhost Hbase]$ vi bulk_data.tsv
```



```
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$ vi bulk_data.tsv
[acadgild@localhost hbase]$ cat bulk_data.tsv
1 AAA 2
2 BBB 3
3 CCC 4
4 DDD 5

You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$
```

Step4:

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



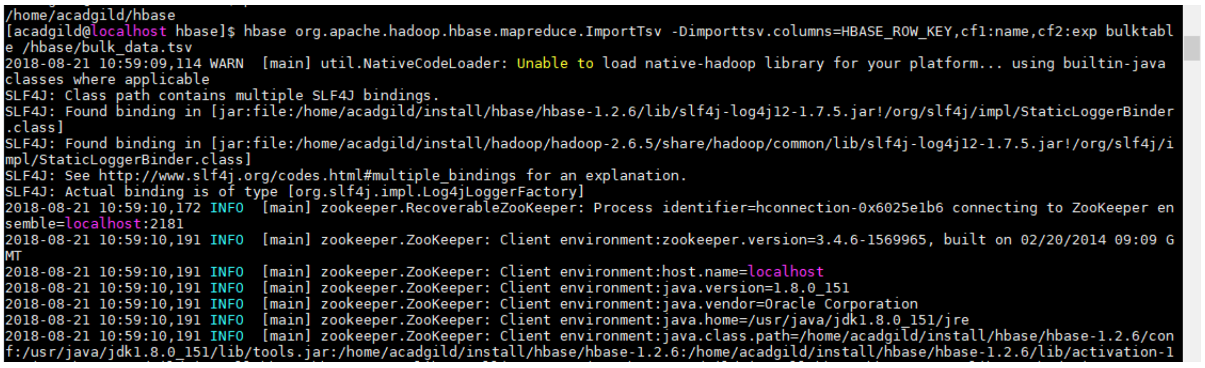
```
[acadgild@localhost hbase]$ ls -l
total 12
-rw-rw-r--. 1 acadgild acadgild  32 Aug 17 12:00 bulk_data1.csv
-rw-rw-r--. 1 acadgild acadgild 302 Aug 17 11:59 bulk_data.csv
-rw-rw-r--. 1 acadgild acadgild  33 Aug 17 12:06 bulk_data.tsv
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$ hadoop fs -put /home/acadgild/Desktop/hbase/bulk_data.tsv /hbase/
18/08/17 12:06:39 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
[acadgild@localhost hbase]$ hadoop fs -cat /hbase/bulk_data.tsv
18/08/17 12:07:03 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
1 AAA 2
2 BBB 3
3 CCC 4
4 DDD 5
```

Step5:

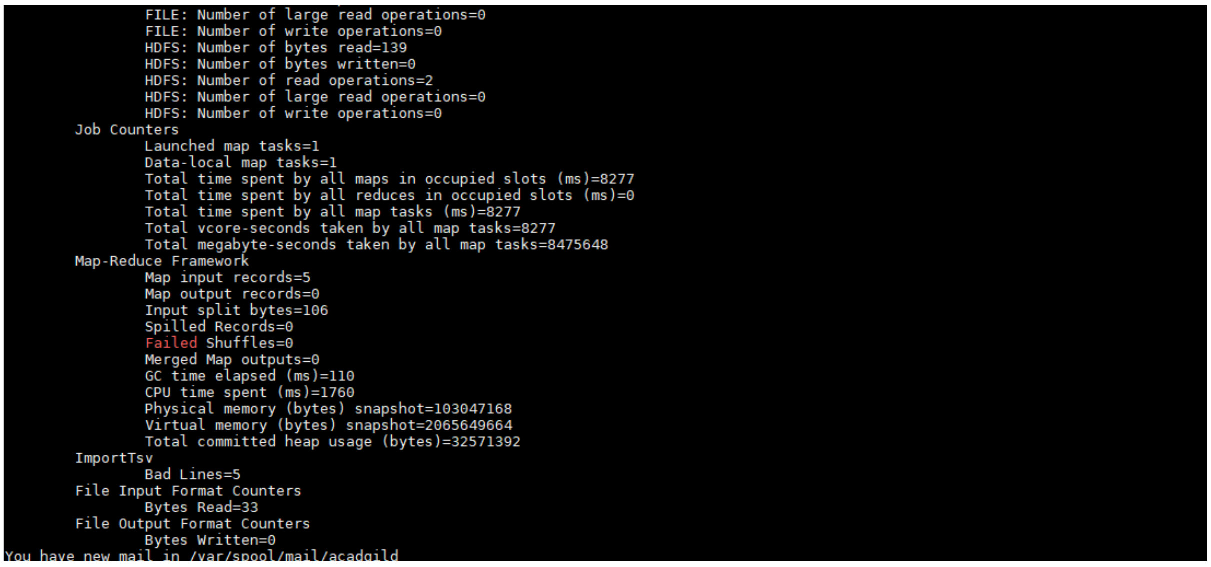
After the data is present now in HDFS.In terminal, we give the following command along with arguments<tablename> and <path of data in HDFS>

Command:

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



```
/home/acadgild/hbase
[acadgild@localhost hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktabl
e /hbase/bulk_data.tsv
2018-08-21 10:59:09,114 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/i
mpl/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]
2018-08-21 10:59:10,172 INFO  [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to ZooKeeper en
semble=localhost:2181
2018-08-21 10:59:10,191 INFO  [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 09:09 G
MT
2018-08-21 10:59:10,191 INFO  [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-08-21 10:59:10,191 INFO  [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-08-21 10:59:10,191 INFO  [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-08-21 10:59:10,191 INFO  [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-08-21 10:59:10,191 INFO  [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hbase-1.2.6/con
f:/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
```



```
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=139
                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)=8277
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=8277
                Total vcore-seconds taken by all map tasks=8277
                Total megabyte-seconds taken by all map tasks=8475648
        Map-Reduce Framework
                Map input records=5
                Map output records=0
                Input split bytes=106
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=110
                CPU time spent (ms)=1760
                Physical memory (bytes) snapshot=103047168
                Virtual memory (bytes) snapshot=2065649664
                Total committed heap usage (bytes)=32571392
        ImportTsv
                Bad Lines=5
        File Input Format Counters
                Bytes Read=33
        File Output Format Counters
                Bytes Written=0
You have new mail in /var/spool/mail/acadgild
```