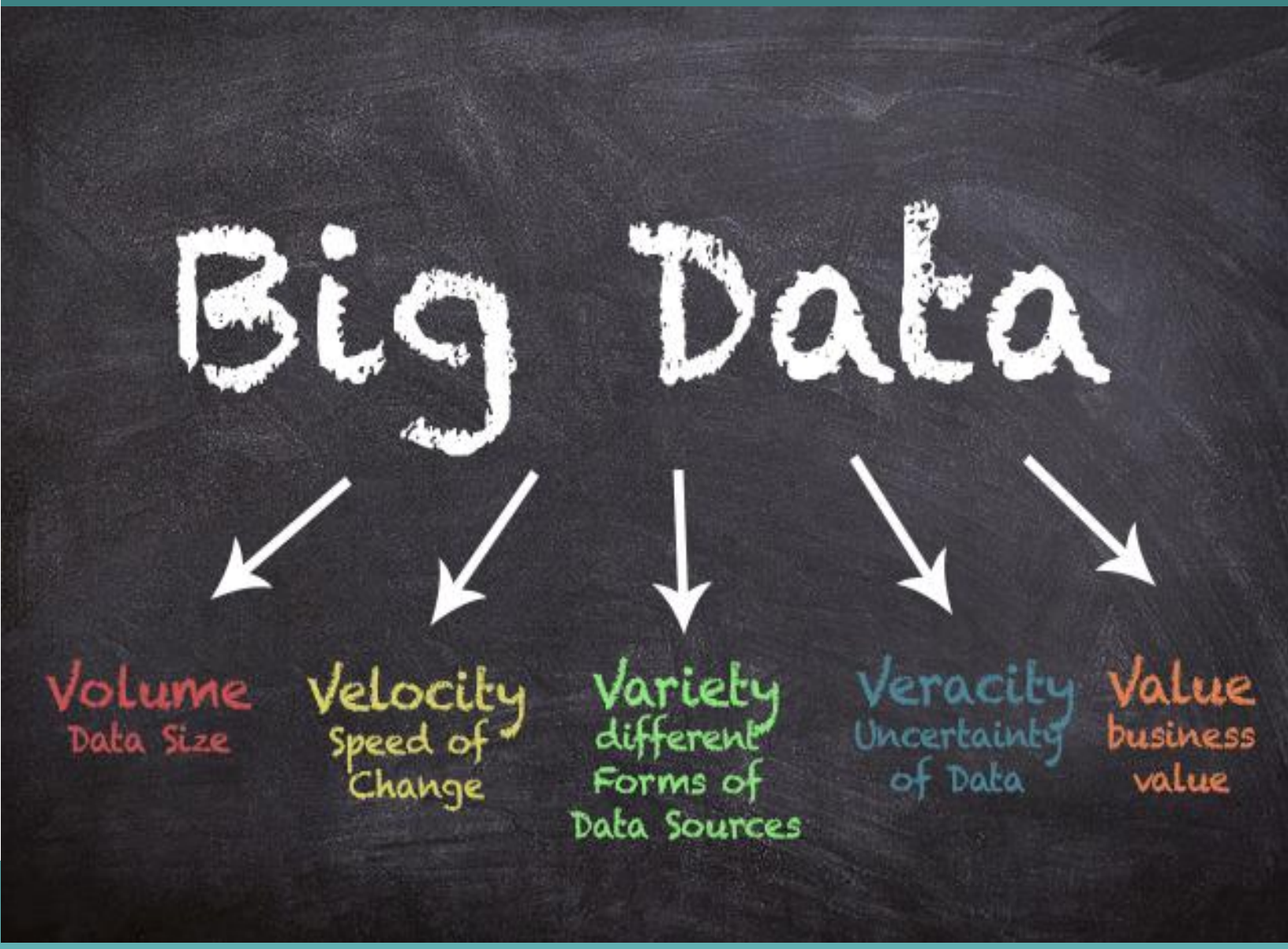
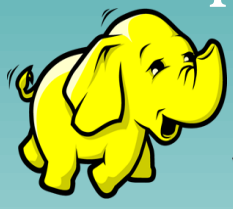
08/09/2018 | By: Ajit K Prasad



Big Data Engineering with Hadoop & Spark

Assignment on Advance HBase



Acadgild

Session 11: Assignment 11.1

This assignment is aimed at consolidating the concepts that was learnt during the Advance HBase session of the course.

Problem Statement

Task 1: Explain the below concepts with an example in brief.

1. 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:

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

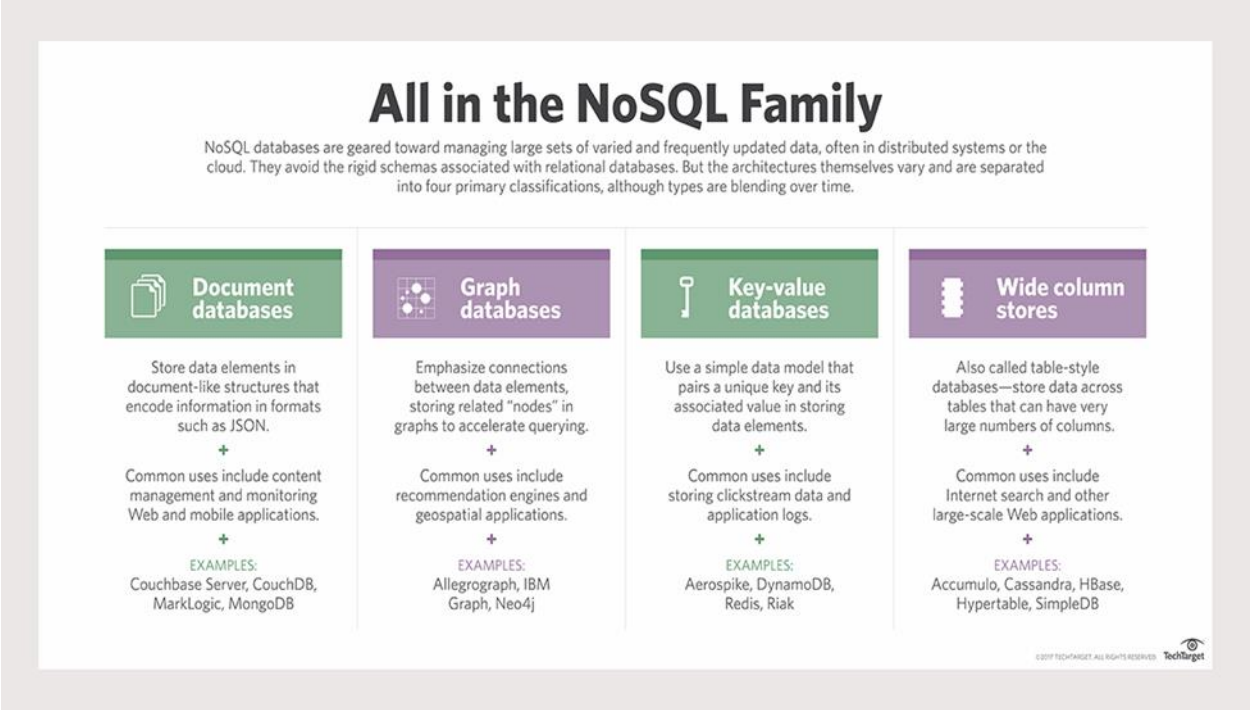
Most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes, so queries for data might not return updated data immediately or might result in reading data that is not accurate, this problem is known as stale reads.

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.

2. 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}



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.

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.

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.

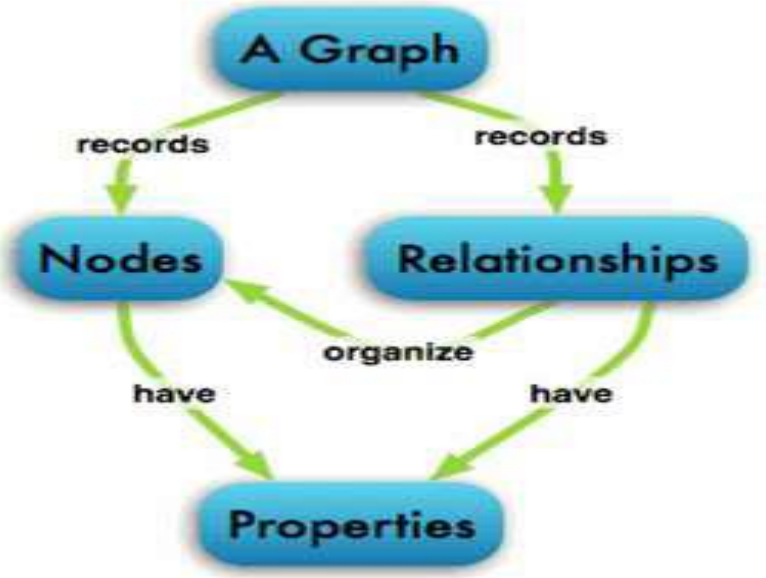
In comparison, most relational DBMS store data in rows, the benefit of storing data in columns, is fast search/ access and data aggregation. Columnar databases store all the cells corresponding to a column as a continuous disk entry thus makes the search/access faster.

For example: To query the titles from articles is just one disk access, title of all the items can be obtained.

Graph Base NoSQL Database

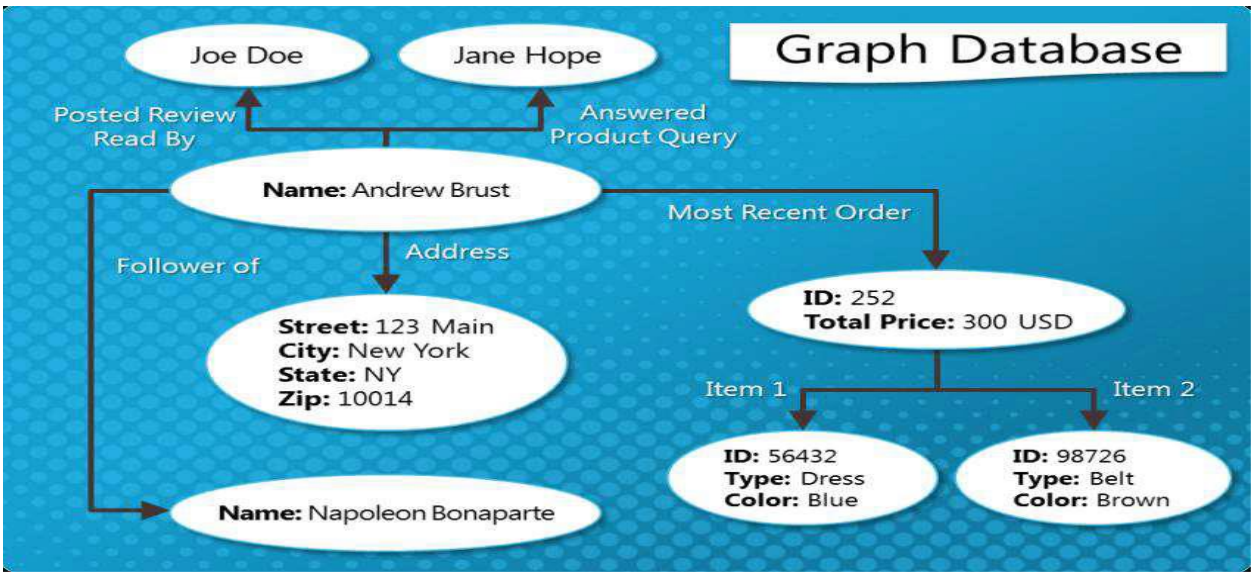
In a Graph Base NoSQL Database, you will not find the rigid format of SQL or the tables and columns representation, a flexible graphical representation is instead used which is perfect to address scalability concerns. Graph structures

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 organized 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.



3. 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 Tolerance: System must work well across distributed networks.

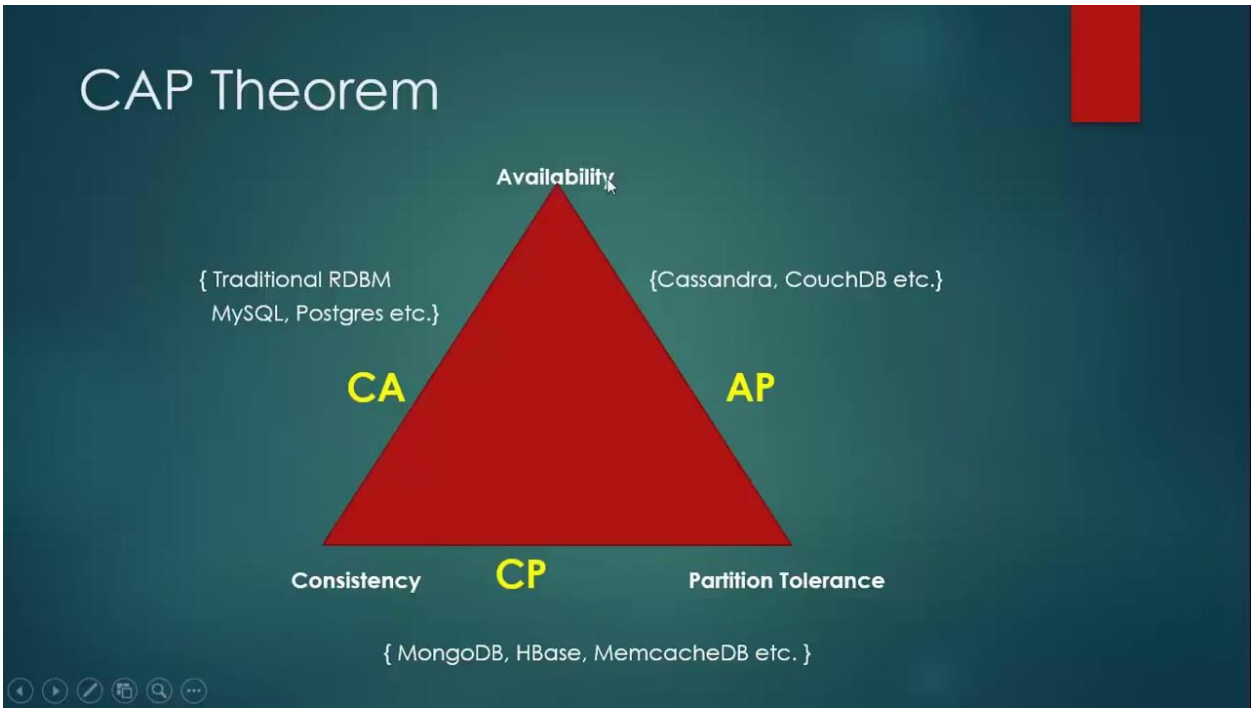
The CAP theorem was proposed by Eric Brewer.

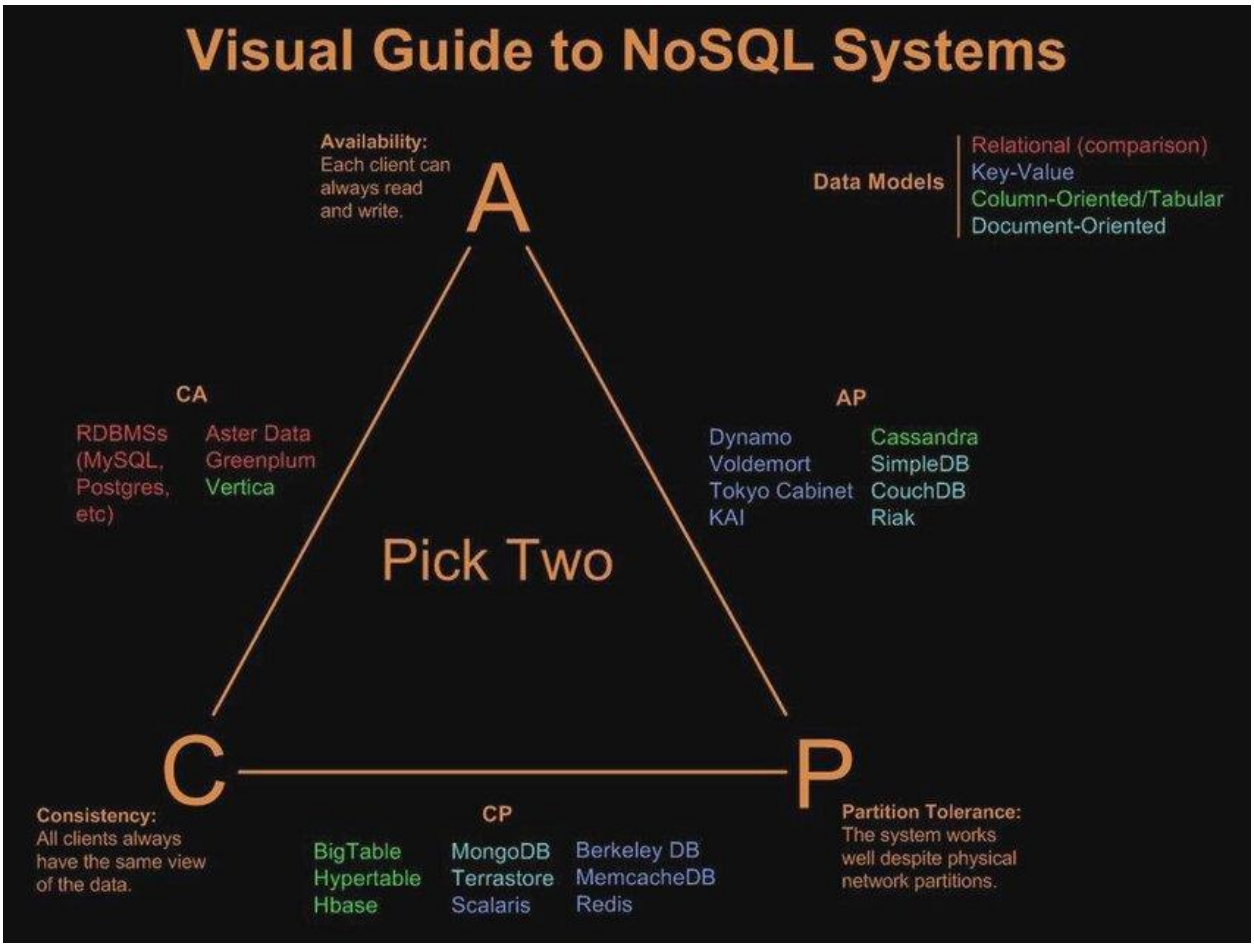
According to this theorem, in any distributed system, you can use only two of the three properties—consistency, availability, or partition tolerance simultaneously.

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.

(Graphical representations of CAP Theorem)





Consistency

- **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.

- **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.

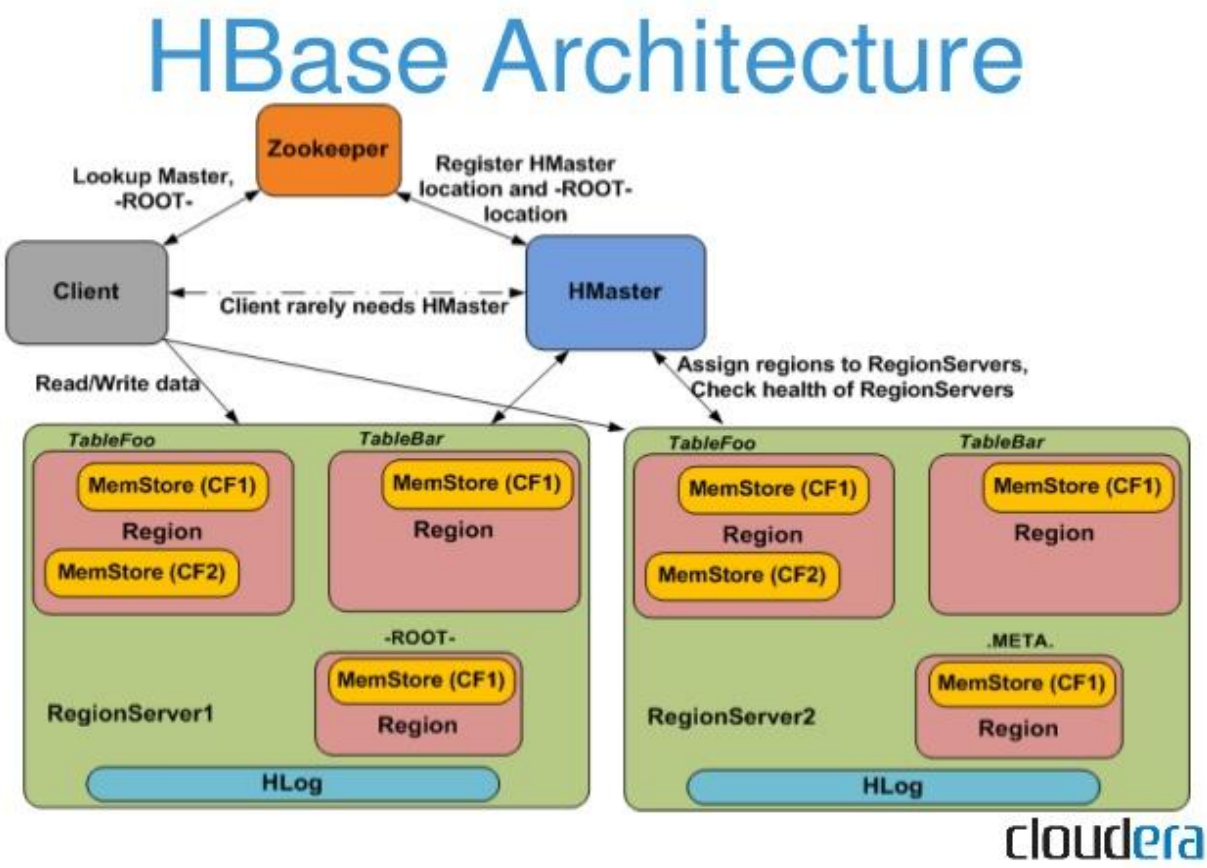
Partition Tolerance

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.

4. HBase Architecture

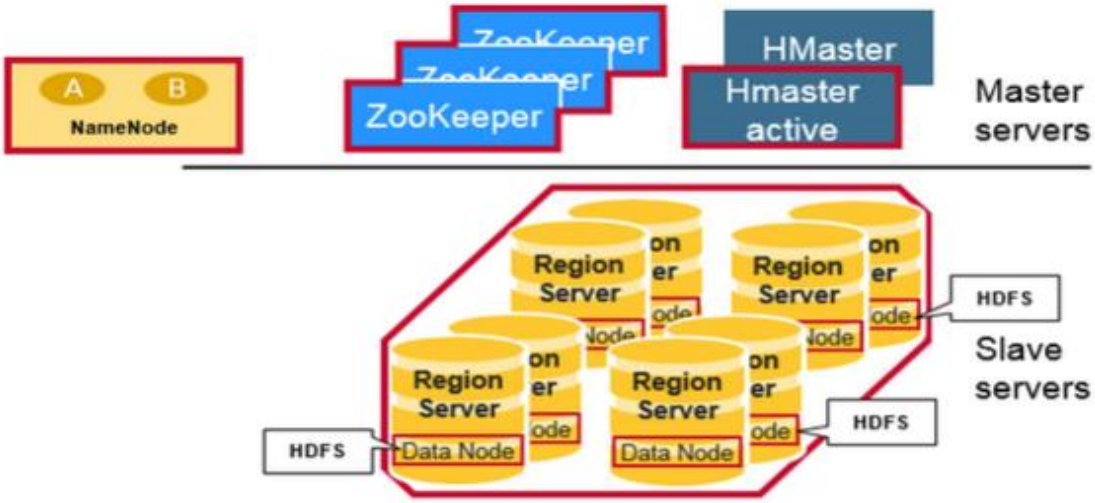
In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into "Stores". Stores are saved as files in HDFS.

Graphical representation of 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.

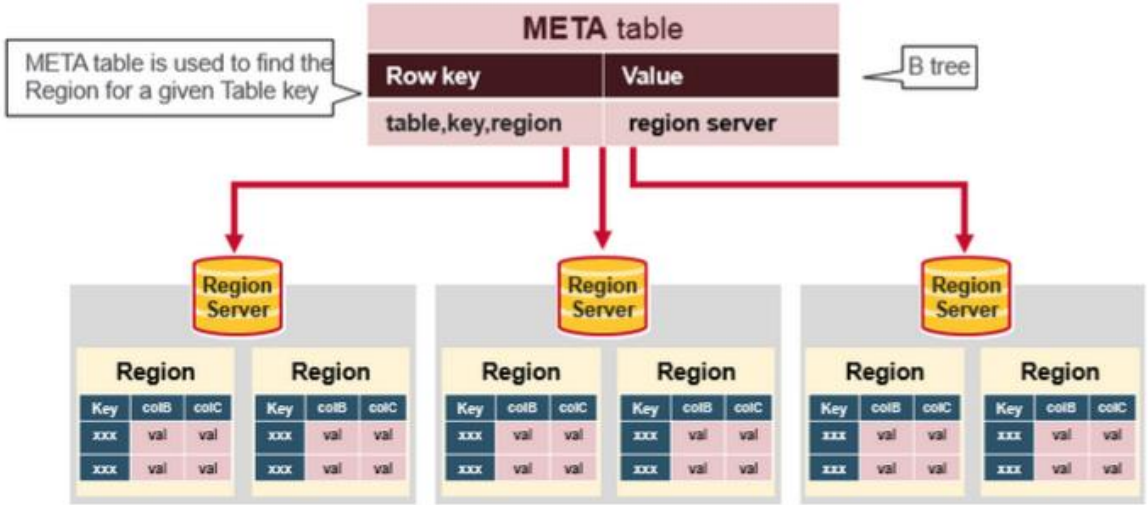
HBase Meta Table:

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:

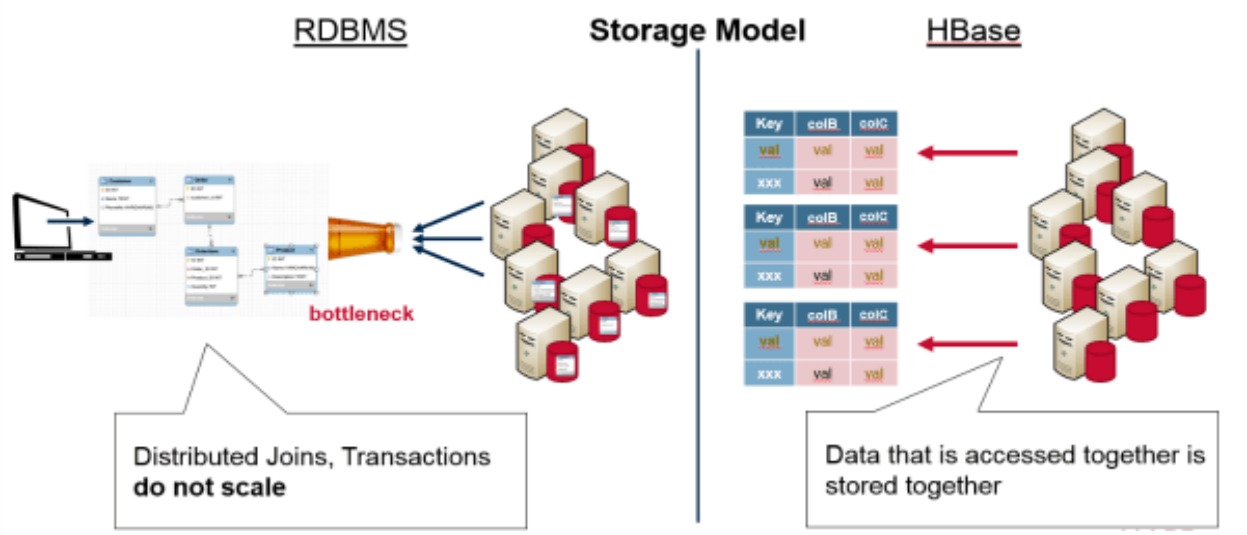
- Key: region start key, region id
- Values: RegionServer

Graphical representation of META Table



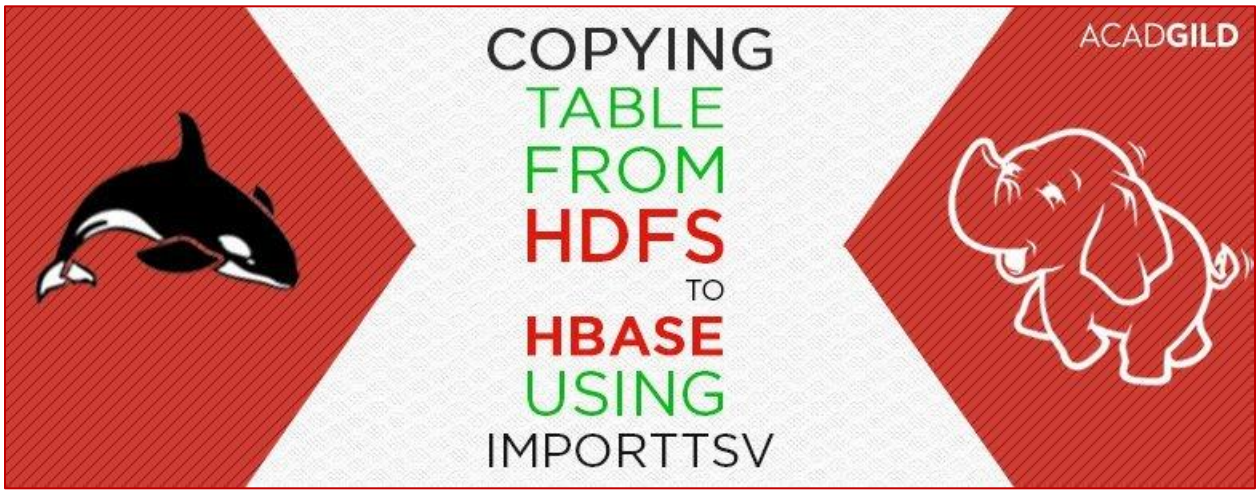
5. HBase vs RDBMS

HBase	RDBMS
Column-oriented	Row-oriented (mostly)
Flexible schema, add columns on the Fly	Fixed schema
Good with sparse tables	Not optimized for sparse tables
No query language	SQL
Wide Tables	Narrow Tables
Joins using MR – not optimized	Optimized for Joins (small, fast ones)
Tight integration with MR	Not really
De-normalize your data	Normalize as you can
Horizontal scalability – just add hardware	Hard to share and scale
Good for semi-structured data as well as structured data	Good for structured data
Atomicity, Consistency, Isolation & Durability (ACID) Compliant	Not ACID Compliant



Task 2:

Execute blog present in below link:
https://acadgild.com/blog/importtsv-data-from-hdfs-into-hbase/



- Start history server, all daemons, hbase daemon and check if the up & running
  - *$ /home/acadgild/install/hadoop/hadoop-2.6.5/sbin/mr-jobhistory-daemon.sh start historyserver*
  - *$ start-all.sh*
  - *$ start-hbase.sh*
  - *$ jps*

```
[acadgild@localhost ~]$ jps
4069 ResourceManager
3753 SecondaryNameNode
3354 NameNode
5227 HRegionServer
6045 Jps
4189 NodeManager
3487 DataNode
3119 JobHistoryServer
4991 HQuorumPeer
5071 HMaster
```

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

  $ hbase shell

  hbase(main):001:0> create 'bulktable', 'cf1', 'cf2'

```
[acadgild@localhost ~]$ hbase shell
2018-08-10 19:00:25,137 WARN  [main] util.NativeCodeLoader
-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hb
Binder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/ha
lf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_binding
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLogge
HBase Shell; enter 'help<RETURN>' for list of supported co
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>
hbase(main):001:0> create 'bulktable', 'cf1', 'cf2'
0 row(s) in 4.0440 seconds

=> Hbase::Table - bulktable
```

- Create a directory in local drive to use it for the process execution and saving the data

  $ mkdir HBase

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

  $ cd Hbase

- Create a file inside the HBase directory named bulk_data.tsv with tab separated data inside using below command in terminal.

  $ vi bulk_data.tsv

- Feed these data in the file

  1 Amit 4
  2 Girija 3
  3 Jatin 5
  4 Swati 3

- Once created, save the file using *esc + :wq + enter*
- Check the data fed on the file created

  $ cat bulk_data.tsv

```
[acadgild@localhost ~]$ mkdir HBase
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ cd HBase
[acadgild@localhost HBase]$ pwd
/home/acadgild/HBase
[acadgild@localhost HBase]$ vi bulk_data.tsv
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost HBase]$ cat bulk_data.tsv
1       Amit    4
2       Girja   3
3       Jatin   5
4       Swati   3
```

– 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 create a directory on HDFS, copy the data inside the directory created and check the data on the file, using below commands in terminal.

$ hadoop fs -mkdir /HBaseExamples
$ hadoop fs -put bulk_data.tsv /HBaseExamples/
$ hadoop fs -cat /HBaseExamples/bulk_data.tsv

```
[acadgild@localhost HBase]$ hadoop fs -mkdir /HBaseExamples
18/08/10 19:06:47 WARN util.NativeCodeLoader: Unable to load native-hadoop li
where applicable
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost HBase]$ hadoop fs -ls /
18/08/10 19:06:58 WARN util.NativeCodeLoader: Unable to load native-hadoop li
where applicable
Found 5 items
drwxr-xr-x   - acadgild supergroup          0 2018-08-10 19:06 /HBaseExamples
drwxr-xr-x   - acadgild supergroup          0 2018-08-10 18:51 /hbase
drwxr-xr-x   - acadgild supergroup          0 2018-08-08 13:39 /home
drwxrwx---   - acadgild supergroup          0 2018-08-08 11:34 /tmp
drwxr-xr-x   - acadgild supergroup          0 2018-08-08 11:35 /user
[acadgild@localhost HBase]$ hadoop fs -put bulk_data.tsv /HBaseExamples/
18/08/10 19:07:51 WARN util.NativeCodeLoader: Unable to load native-hadoop li
where applicable
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost HBase]$ hadoop fs -cat /HBaseExamples/bulk_data.tsv
18/08/10 19:08:28 WARN util.NativeCodeLoader: Unable to load native-hadoop li
where applicable
1       Amit    4
2       Girja   3
3       Jatin   5
4       Swati   3
```

– After the data is present now in HDFS. On local 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 /HBaseExamples/bulk_data.tsv

```
[acadgild@localhost HBase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bu
lktable /HBaseExamples/bulk_data.tsv
2018-08-10 19:13:04,379 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/StaticLogger
Binder.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/s
lf4j/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]
2018-08-10 19:13:06,126 INFO  [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to ZooKee
per ensemble=localhost:2181
2018-08-10 19:13:06,176 INFO  [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/2014 0
9:09 GMT
2018-08-10 19:13:06,177 INFO  [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-08-10 19:13:06,177 INFO  [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-08-10 19:13:06,177 INFO  [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-08-10 19:13:06,177 INFO  [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-08-10 19:13:06,178 INFO  [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hbase-1.2
2018-08-10 19:13:16,862 INFO  [main] mapreduce.JobSubmitter: number of splits:1
2018-08-10 19:13:16,893 INFO  [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-che
cksum
2018-08-10 19:13:17,631 INFO  [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1533907238622_0001
2018-08-10 19:13:19,130 INFO  [main] impl.YarnClientImpl: Submitted application application_1533907238622_0001
2018-08-10 19:13:19,510 INFO  [main] mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1533907238622
_0001/
2018-08-10 19:13:19,512 INFO  [main] mapreduce.Job: Running job: job_1533907238622_0001
2018-08-10 19:13:47,439 INFO  [main] mapreduce.Job: Job job_1533907238622_0001 running in uber mode : false
2018-08-10 19:13:47,454 INFO  [main] mapreduce.Job:  map 0% reduce 0%
2018-08-10 19:14:04,987 INFO  [main] mapreduce.Job:  map 100% reduce 0%
2018-08-10 19:14:07,155 INFO  [main] mapreduce.Job: Job job_1533907238622_0001 completed successfully
2018-08-10 19:14:07,492 INFO  [main] mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=139471
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=153
                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)=14942
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=14942
                Total vcore-seconds taken by all map tasks=14942
                Total megabyte-seconds taken by all map tasks=15300608
        Map-Reduce Framework
                Map input records=4
                Map output records=4
                Input split bytes=114
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=375
                CPU time spent (ms)=5660
                Physical memory (bytes) snapshot=183271424
                Virtual memory (bytes) snapshot=2099679232
                Total committed heap usage (bytes)=100139008
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=39
        File Output Format Counters
                Bytes Written=0
You have new mail in /var/spool/mail/acadgild
```

- Observe that the map is done 100% although we get an error afterward. For now, ignore the error message as our task is to map data in HBase table.
- Now, also let us check whether we actually got the data inside HBase by using the command on HBase Shell

  hbase(main):002:0> scan 'bulktable'

```
hbase(main):002:0> scan 'bulktable'
ROW                                   COLUMN+CELL
 1                                    column=cf1:name, timestamp=1533908584263, value=Amit
 1                                    column=cf2:exp, timestamp=1533908584263, value=4
 2                                    column=cf1:name, timestamp=1533908584263, value=Girja
 2                                    column=cf2:exp, timestamp=1533908584263, value=3
 3                                    column=cf1:name, timestamp=1533908584263, value=Jatin
 3                                    column=cf2:exp, timestamp=1533908584263, value=5
 4                                    column=cf1:name, timestamp=1533908584263, value=Swati
 4                                    column=cf2:exp, timestamp=1533908584263, value=3
4 row(s) in 0.5740 seconds
```

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

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
  Other options that may be specified with -D include:
  - **-Dimporttsv.skip.bad.lines=false** – fail if encountering an invalid line
  - **'-Dimporttsv.separator=|'** – e.g., 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**