

Assignment 11: Advance Hbase

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

● Nosql Databases

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.

Example: Hbase, NoSQL.

● Types of Nosql Databases

There are various types of Nosql databases are

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 a concept that a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance.

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. I will try to explain each of these concepts and the reasons for the trade off. I will avoid using specific examples as DBMS are rapidly evolving.

Partition Tolerance

This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes. A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network. Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent

outages. When dealing with modern distributed systems, Partition Tolerance is not an option. It's a necessity. Hence, we have to trade between Consistency and Availability.

High Consistency

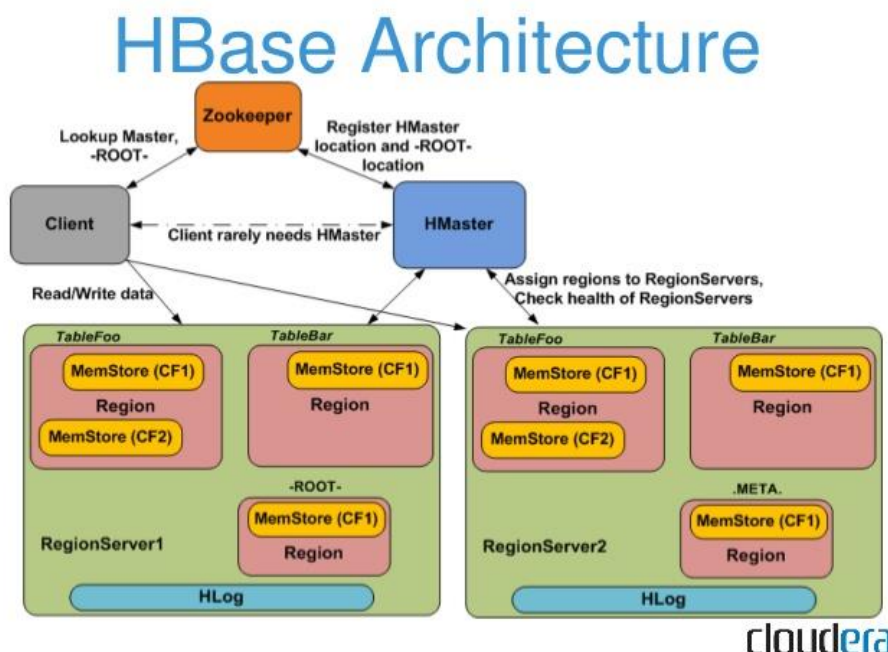
This condition states that all nodes see the same data at the same time. Simply put, performing a read operation will return the value of the most recent write operation causing all nodes to return the same data. A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state.

High Availability

This condition states that every request gets a response on success/failure. Achieving availability in a distributed system requires that the system remains operational 100% of the time. Every client gets a response, regardless of the state of any individual node in the system. This metric is trivial to measure: either you can submit read/write commands, or you cannot. Hence, the databases are time independent as the nodes need to be available online at all times.

● HBase Architecture

HBase provides low-latency random reads and writes on top of HDFS. In HBase, tables are dynamically distributed by the system whenever they become too large to handle (Auto Sharding). The simplest and foundational unit of horizontal scalability in HBase is a Region. A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data). HBase architecture has a single HBase master node (HMaster) and several slaves i.e. region servers. Each region server (slave) serves a set of regions, and a region can be served only by a single region server. Whenever a client sends a write request, HMaster receives the request and forwards it to the corresponding region server.



HBase can be run in a multiple master setup, wherein there is only single active master at a time. HBase tables are partitioned into multiple regions with every region storing multiple table's rows.

Components of Apache HBase Architecture

HBase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

HMaster

HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

- Manages and Monitors the Hadoop Cluster
- Performs Administration (Interface for creating, updating and deleting tables.)
- Controlling the failover
- DDL operations are handled by the HMaster

Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

Region Server

These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

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

Zookeeper

HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning. ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization. Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.

ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

- Establishing client communication with region servers.
- Tracking server failure and network partitions.
- Maintain Configuration Information
- Provides ephemeral nodes, which represent different region servers.

● **HBase vs RDBMS**

Hbase	RDBMS
Column Oriented.	Row Oriented(Mostly).
Flexible Schema, add columns on the fly.	Fixed Schema.
Good with Sparse Table.	Not optimized with sparse table.

Joins using MR- not Optimized.	Optimized for joins.
Tight integration with MR.	Not really.
Horizontal scalability – just to add hardware.	Hard to shard and scale.
Good for Semi-structured as well as un-structured data.	Good for Structured data.

Task 2: Import TSV Data from HDFS into HBase

Solution: Step 1: Inside Hbase shell give the following command to create table along with 2 column family.

create 'bulktable','cf1','cf2'

```

hbase(main):007:0> create 'bulktable','cf1','cf2'
0 row(s) in 1.2670 seconds

=> Hbase::Table - bulktable
hbase(main):008:0> list
TABLE
bulktable
1 row(s) in 0.0080 seconds

=> ["bulktable"]
hbase(main):009:0> describe 'bulktable'
Table bulktable is ENABLED
bulktable
COLUMN FAMILIES DESCRIPTION
(NAME => 'cf1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true')
(NAME => 'cf2', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true')
2 row(s) in 0.0670 seconds

hbase(main):010:0> scan 'bulktable'
ROW
0 row(s) in 0.0580 seconds

hbase(main):011:0>

```

Step 2: Come out of HBase shell to the terminal and also make a directory for Hbase in the local drive; So, since we have our own path you can use it.

mkdir -p Hbase

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

cd Hbase

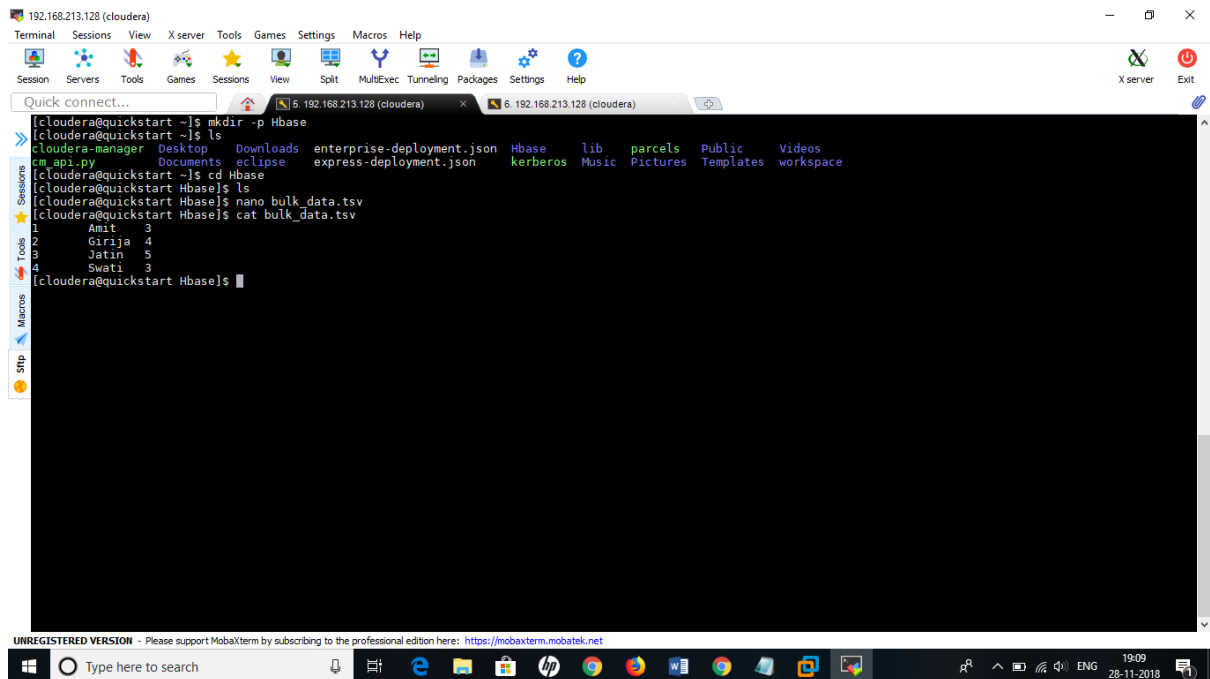
Step 3: Create a file inside the HBase directory named bulk_data.tsv with tab separated data inside using below command in terminal.

nano bulk_data.tsv

Put these data in,

- 1 Amit 4
- 2 Girija 3

- 3 Jatin 5
- 4 Swati 3



```
[cloudera@quickstart ~]$ mkdir -p Hbase
[cloudera@quickstart ~]$ ls
cm_api.py  Desktop  Downloads  enterprise-deployment.json  Hbase  lib  parcels  Public  Videos
[cloudera@quickstart ~]$ cd Hbase
[cloudera@quickstart Hbase]$ ls
[cloudera@quickstart Hbase]$ nano bulk_data.tsv
[cloudera@quickstart Hbase]$ cat bulk_data.tsv
1      Amit      3
2      Girija   4
3      Jatin    5
4      Swati    3
[cloudera@quickstart Hbase]$
```

Step 4: Our data should be present in HDFS while performing the import task to Hbase.

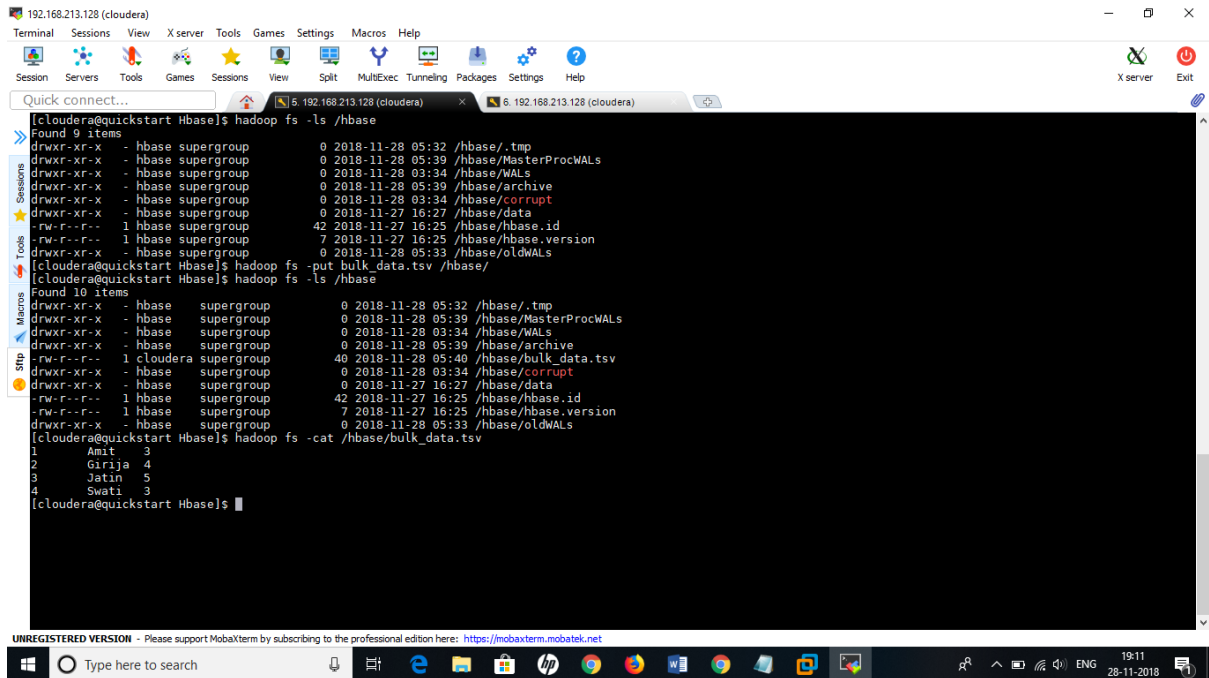
In real time projects, the data will already be present inside HDFS.

Here for our learning purpose, we copy the data inside HDFS using below commands in terminal.

Command: *hadoop fs -mkdir /hbase*

hadoop fs -put bulk_data.tsv/hbase/

hadoop fs -cat/hbase/bulk_data.tsv

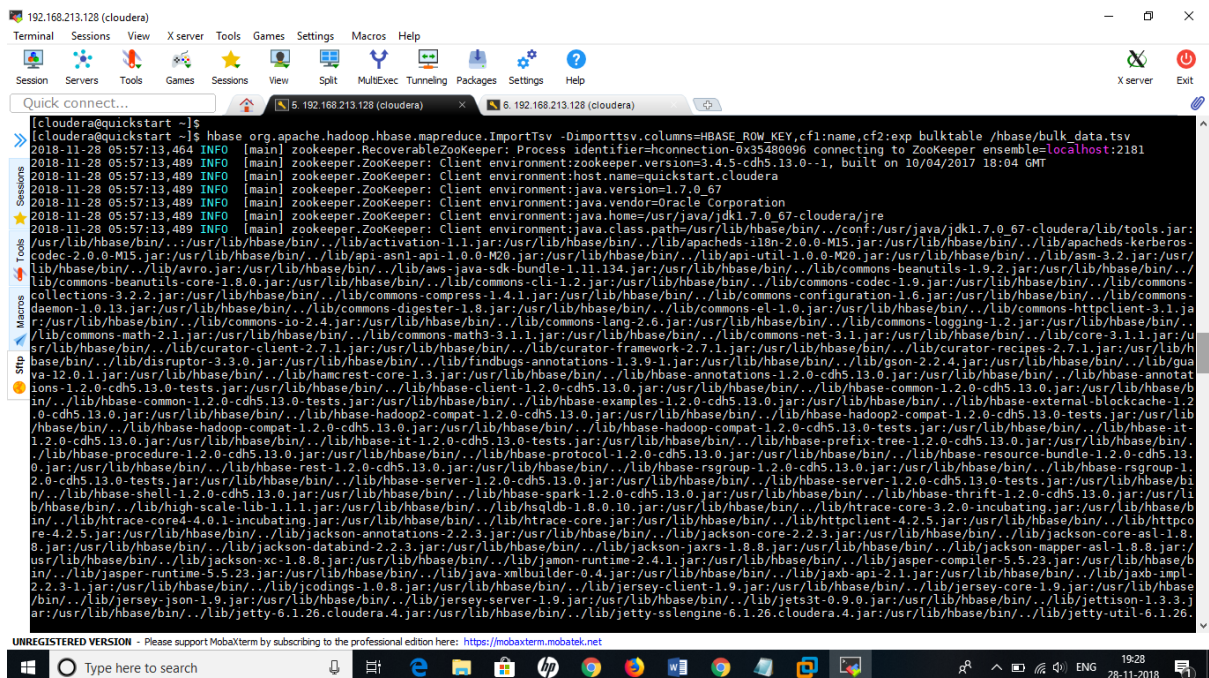


```
[cloudera@quickstart Hbase]$ hadoop fs -ls /hbase
Found 9 items
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:32 /hbase/.tmp
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:39 /hbase/MasterProcWALS
drwxr-xr-x - hbase supergroup 0 2018-11-28 03:34 /hbase/WALs
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:39 /hbase/archive
drwxr-xr-x - hbase supergroup 0 2018-11-28 03:34 /hbase/corrupt
drwxr-xr-x - hbase supergroup 0 2018-11-27 16:27 /hbase/data
-rw-r--r-- 1 hbase supergroup 42 2018-11-27 16:25 /hbase/hbase.id
-rw-r--r-- 1 hbase supergroup 7 2018-11-27 16:25 /hbase/hbase.version
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:33 /hbase/oldWALS
[cloudera@quickstart Hbase]$ hadoop fs -put bulk_data.tsv /hbase/
[cloudera@quickstart Hbase]$ hadoop fs -ls /hbase
Found 10 items
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:32 /hbase/.tmp
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:39 /hbase/MasterProcWALS
drwxr-xr-x - hbase supergroup 0 2018-11-28 03:34 /hbase/WALs
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:39 /hbase/archive
-rw-r--r-- 1 cloudera supergroup 40 2018-11-28 05:40 /hbase/bulk_data.tsv
drwxr-xr-x - hbase supergroup 0 2018-11-28 03:34 /hbase/corrupt
drwxr-xr-x - hbase supergroup 0 2018-11-27 16:27 /hbase/data
-rw-r--r-- 1 hbase supergroup 42 2018-11-27 16:25 /hbase/hbase.id
-rw-r--r-- 1 hbase supergroup 7 2018-11-27 16:25 /hbase/hbase.version
drwxr-xr-x - hbase supergroup 0 2018-11-28 05:33 /hbase/oldWALS
[cloudera@quickstart Hbase]$ hadoop fs -cat /hbase/bulk_data.tsv
Amit 3
Girija 4
Jatin 5
Swati 3
[cloudera@quickstart Hbase]$
```

Step 5: After the data is present now in HDFS. In terminal, we give the following command along with arguments <tablename> and <path of data in HDFS>

Command:

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



```
[cloudera@quickstart ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv
2018-11-28 05:57:13,464 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x35480096 connecting to ZooKeeper ensemble=localhost:2181
2018-11-28 05:57:13,489 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.5-cdh5.13.0-1, built on 10/04/2017 18:04 GMT
2018-11-28 05:57:13,489 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=quickstart.cloudera
2018-11-28 05:57:13,489 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.7.0_67
2018-11-28 05:57:13,489 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-11-28 05:57:13,489 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.7.0_67-cloudera/lib
2018-11-28 05:57:13,489 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=/usr/lib/hbase/bin/./conf:/usr/java/jdk1.7.0_67-cloudera/lib/tools.jar:/usr/lib/hbase/bin/./lib/activation-1.1.jar:/usr/lib/hbase/bin/./lib/apacheds-lln-2.0.0-M15.jar:/usr/lib/hbase/bin/./lib/apacheds-kerberos-codec-2.0.0-M15.jar:/usr/lib/hbase/bin/./lib/api-asn1-api-1.0.0-M20.jar:/usr/lib/hbase/bin/./lib/api-util-1.0.0-M20.jar:/usr/lib/hbase/bin/./lib/asm-3.2.jar:/usr/lib/hbase/bin/./lib/avro.jar:/usr/lib/hbase/bin/./lib/aws-java-sdk-bundle-1.11.134.jar:/usr/lib/hbase/bin/./lib/commons-beanutils-1.9.2.jar:/usr/lib/hbase/bin/./lib/commons-beanutils-core-1.8.0.jar:/usr/lib/hbase/bin/./lib/commons-cli-1.2.jar:/usr/lib/hbase/bin/./lib/commons-codec-1.9.jar:/usr/lib/hbase/bin/./lib/commons-collections-3.2.2.jar:/usr/lib/hbase/bin/./lib/commons-compress-1.4.1.jar:/usr/lib/hbase/bin/./lib/commons-configuration-1.6.jar:/usr/lib/hbase/bin/./lib/commons-daemon-1.0.13.jar:/usr/lib/hbase/bin/./lib/commons-digester-1.8.jar:/usr/lib/hbase/bin/./lib/commons-el-1.0.jar:/usr/lib/hbase/bin/./lib/commons-httpclient-3.1.jar:/usr/lib/hbase/bin/./lib/commons-io-2.4.jar:/usr/lib/hbase/bin/./lib/commons-lang-2.6.jar:/usr/lib/hbase/bin/./lib/commons-logging-1.2.jar:/usr/lib/hbase/bin/./lib/commons-math-2.1.jar:/usr/lib/hbase/bin/./lib/commons-math3-3.1.1.jar:/usr/lib/hbase/bin/./lib/commons-net-3.1.jar:/usr/lib/hbase/bin/./lib/core-3.1.1.jar:/usr/lib/hbase/bin/./lib/curator-client-2.7.1.jar:/usr/lib/hbase/bin/./lib/curator-framework-2.7.1.jar:/usr/lib/hbase/bin/./lib/curator-recipes-2.7.1.jar:/usr/lib/hbase/bin/./lib/disruptor-3.3.0.jar:/usr/lib/hbase/bin/./lib/findbugs-annotations-1.3.9.1.jar:/usr/lib/hbase/bin/./lib/gson-2.2.4.jar:/usr/lib/hbase/bin/./lib/guava-12.0.1.jar:/usr/lib/hbase/bin/./lib/hamcrest-core-1.3.jar:/usr/lib/hbase/bin/./lib/hbase-annotations-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-annotations-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-client-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-client-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-common-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-common-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-examples-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-external-blockcache-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-hadoop2-compat-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-hadoop2-compat-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-hadoop-compat-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-hadoop-compat-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-it-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-it-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-prefix-tree-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-procedure-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-protocol-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-resource-bundle-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-rest-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-rsgroup-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-rsgroup-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-server-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-server-1.2.0-cdh5.13.0-tests.jar:/usr/lib/hbase/bin/./lib/hbase-shell-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-spark-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-thrift-1.2.0-cdh5.13.0.jar:/usr/lib/hbase/bin/./lib/hbase-trace-core-4.0.1-incubating.jar:/usr/lib/hbase/bin/./lib/hbase-trace-core.jar:/usr/lib/hbase/bin/./lib/hbase-trace-core-4.0.1-incubating.jar:/usr/lib/hbase/bin/./lib/hbase-trace-core.jar:/usr/lib/hbase/bin/./lib/httpclient-4.2.5.jar:/usr/lib/hbase/bin/./lib/httpcore-4.2.5.jar:/usr/lib/hbase/bin/./lib/jackson-annotations-2.2.3.jar:/usr/lib/hbase/bin/./lib/jackson-core-2.2.3.jar:/usr/lib/hbase/bin/./lib/jackson-core-asl-1.8.8.jar:/usr/lib/hbase/bin/./lib/jackson-databind-2.2.3.jar:/usr/lib/hbase/bin/./lib/jackson-jaxrs-1.8.8.jar:/usr/lib/hbase/bin/./lib/jackson-mapper-asl-1.8.8.jar:/usr/lib/hbase/bin/./lib/jackson-rx-1.8.8.jar:/usr/lib/hbase/bin/./lib/jamon-runtime-2.4.1.jar:/usr/lib/hbase/bin/./lib/jasper-compiler-5.5.23.jar:/usr/lib/hbase/bin/./lib/jasper-runtime-5.5.23.jar:/usr/lib/hbase/bin/./lib/java-xmlbuilder-0.4.jar:/usr/lib/hbase/bin/./lib/jaxb-api-2.1.jar:/usr/lib/hbase/bin/./lib/jaxb-impl-2.2.3-1.jar:/usr/lib/hbase/bin/./lib/jcodings-1.0.8.jar:/usr/lib/hbase/bin/./lib/jersey-client-1.9.jar:/usr/lib/hbase/bin/./lib/jersey-core-1.9.jar:/usr/lib/hbase/bin/./lib/jersey-json-1.9.jar:/usr/lib/hbase/bin/./lib/jersey-server-1.9.jar:/usr/lib/hbase/bin/./lib/jets3t-0.9.0.jar:/usr/lib/hbase/bin/./lib/jettison-1.3.3.jar:/usr/lib/hbase/bin/./lib/jetty-6.1.26.cloudera.4.jar:/usr/lib/hbase/bin/./lib/jetty-sslengine-6.1.26.cloudera.4.jar:/usr/lib/hbase/bin/./lib/jetty-util-6.1.26.cloudera.4.jar
```

```
192.168.213.128 (cloudera)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
5. 192.168.213.128 (cloudera)
6. 192.168.213.128 (cloudera)
2018-11-28 05:57:17,841 INFO [main-EventThread] zookeeper.ClientCnxn: EventThread shut down
2018-11-28 05:57:17,841 INFO [main-EventThread] ZooKeeper: Session: 0x1675a1463ab000e closed
2018-11-28 05:57:18,182 INFO [main] client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2018-11-28 05:57:18,691 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-11-28 05:57:22,136 WARN [main] DataStreamer for file /tmp/hadoop-yarn/staging/cloudera/.staging/job_1543404658981_0001/libjars/protobuf-java-2.5.0.jar] hdfs.DFSClient: Caught exception
java.lang.InterruptedExecutionException
    at java.lang.Object.wait(Native Method)
    at java.lang.Thread.join(Thread.java:1281)
    at java.lang.Thread.join(Thread.java:1355)
    at org.apache.hadoop.hdfs.DFSOutputStreamDataStreamer.closeResponder(DFSOutputStream.java:967)
    at org.apache.hadoop.hdfs.DFSOutputStreamDataStreamer.endBlock(DFSOutputStream.java:705)
    at org.apache.hadoop.hdfs.DFSOutputStreamDataStreamer.run(DFSOutputStream.java:894)
2018-11-28 05:57:22,308 INFO [main] input.FileInputFormat: Total input paths to process : 1
2018-11-28 05:57:22,503 WARN [main] DataStreamer for file /tmp/hadoop-yarn/staging/cloudera/.staging/job_1543404658981_0001/job.split] hdfs.DFSClient: Caught exception
java.lang.InterruptedExecutionException
    at java.lang.Object.wait(Native Method)
    at java.lang.Thread.join(Thread.java:1281)
    at java.lang.Thread.join(Thread.java:1355)
    at org.apache.hadoop.hdfs.DFSOutputStreamDataStreamer.closeResponder(DFSOutputStream.java:967)
    at org.apache.hadoop.hdfs.DFSOutputStreamDataStreamer.endBlock(DFSOutputStream.java:705)
    at org.apache.hadoop.hdfs.DFSOutputStreamDataStreamer.run(DFSOutputStream.java:894)
2018-11-28 05:57:22,560 INFO [main] mapreduce.JobSubmitter: number of splits:1
2018-11-28 05:57:22,621 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2018-11-28 05:57:23,161 INFO [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1543404658981_0001
2018-11-28 05:57:24,554 INFO [main] impl.YarnClientImpl: Submitted application application_1543404658981_0001
2018-11-28 05:57:24,747 INFO [main] mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1543404658981_0001/
2018-11-28 05:57:24,751 INFO [main] mapreduce.Job: Running job: job_1543404658981_0001
2018-11-28 05:57:44,439 INFO [main] mapreduce.Job: Job job_1543404658981_0001 running in uber mode : false
2018-11-28 05:57:44,441 INFO [main] mapreduce.Job: map 0% reduce 0%
2018-11-28 05:58:02,206 INFO [main] mapreduce.Job: map 100% reduce 0%
2018-11-28 05:58:04,247 INFO [main] mapreduce.Job: Job job_1543404658981_0001 completed successfully
2018-11-28 05:58:04,813 INFO [main] mapreduce.Job: Counters: 31
File System Counters
  FILE: Number of bytes read=0
  FILE: Number of bytes written=188780
  FILE: Number of read operations=0
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```

Scan 'bulkdata'

```
192.168.213.128 (cloudera)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
5. 192.168.213.128 (cloudera)
6. 192.168.213.128 (cloudera)
hbase(main):007:0> create 'bulktable','cf1','cf2'
0 row(s) in 1.2670 seconds

=> hbase::Table - bulktable
hbase(main):008:0> list
TABLE
bulktable
1 row(s) in 0.0080 seconds

=> ["bulktable"]
hbase(main):009:0> describe 'bulktable'
Table bulktable is ENABLED
bulktable
COLUMN FAMILIES DESCRIPTION
(NAME => 'cf1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true')
(NAME => 'cf2', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true')
2 row(s) in 0.0670 seconds

hbase(main):010:0> scan 'bulktable'
COLUMN+CELL
0 row(s) in 0.0580 seconds

hbase(main):011:0> scan 'bulktable'
COLUMN+CELL
1
column=cf1:name, timestamp=1543413430898, value=Amit
1
column=cf2:exp, timestamp=1543413430898, value=3
2
column=cf1:name, timestamp=1543413430898, value=Girija
2
column=cf2:exp, timestamp=1543413430898, value=4
3
column=cf1:name, timestamp=1543413430898, value=Jatin
3
column=cf2:exp, timestamp=1543413430898, value=5
4
column=cf1:name, timestamp=1543413430898, value=Swati
4
column=cf2:exp, timestamp=1543413430898, value=3
4 row(s) in 0.0760 seconds

hbase(main):012:0> █
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```