# NoSǪL-CFDB-HBase-3

Complete the following mini project

Mini Project: Loading and retrieving large-size data using HiveǪL with/without compression In this mini project, we will check the effectiveness of compression in HBase using a compression algorithm. You will load large-size data into HBase, retrieve data from HBase using HiveǪL, and evaluate the effectiveness of compression techniques in HBase. To do so, you will do some tasks like the below:

* Create two tables in HBase
* Load the dataset into HBase tables
* Create two tables in Hive to access the table in your HBase database
* Retrieve data from HBase and evaluate the performance Dataset We are using a larger tweet dataset, which was collected by a tweet collection program, yourTwapperKeeper(ytk). It consists of 12 different collections and has about 5 million tweets. To specify columns in the ImportTsv program, column family name and column (qualifier) names are listed below:

HBASE\_ROW\_KEY,tweet:colnum,tweet:archivesource,tweet:text,tweet:cleantext,tweet:to

\_us er\_id,tweet:from\_user,tweet:tweet\_id,tweet:from\_user\_id,tweet:iso\_language\_code,tweet

:source,tweet:profile\_image\_url,tweet:geo\_type,tweet:geo\_coordinates\_0,tweet:geo\_coor dinates\_1,tweet:created\_at,tweet:time

The dataset (ytk5m.csv, 2GB in total, 4.9 million rows’ tweets table) ALREADY uploaded under /user/data/CSC561NoSǪL/ in HDFS. Note: You SHOULD NOT copy it to your user directory in HDFS or home directory in the Linux filesystem. To load the dataset into HBase, first you need to create two tables in HBase: a table with no compression and another table with compression option (algorithm: Snappy). When you load the dataset, use the above schema for the tables in HBase and use ‘string’ type for all columns. One loading will take more than 5 minutes due to large size dataset (2GB, 4.9 million rows).

----------------------------------------------------------------------------------------------------

Write codes for the following assignments. Take all screenshots of running your codes and

outputs.

Note: Test all your codes first, then take screenshots with NO ERRORS in the screenshots (You

may get fewer points if you have error messages in your screenshots. Tip: Text file editor may

help you to record/edit/save your codes)

Note2: If your output is quite long due to MapReduce job logs, take screenshots of important

parts only without all job logs, e.g., first executing part and last output part.

* Create two tables with and without compression options in HBase
* Use the following filenames:

§ ‘ytk5m’ for uncompressed one

§ ‘ytk5m\_c’ for compressed one

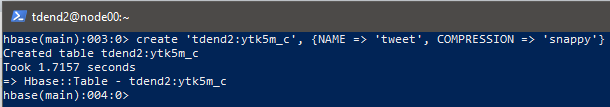
* Compression algorithm: Snappy

Note: Use your database/schema/namespace/login as a prefix of your table names. Note: Column family name: tweet.

# Execution:

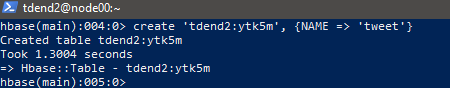
**# Create a table with compression**

create 'tdend2:ytk5m\_c', {NAME => 'tweet', COMPRESSION => 'snappy'}



# # Create a table with compression ‘none’

create 'tdend2:ytk5m', {NAME => 'tweet'}

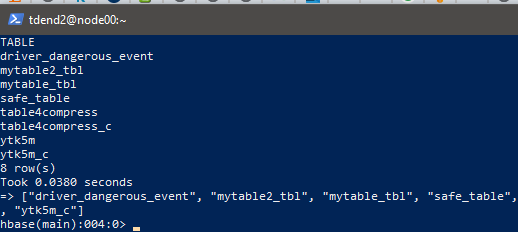


-

* List your tables using an HBase shell command ONLY. (NOT other ways)

# Execution:

list\_namespace\_tables 'tdend2'



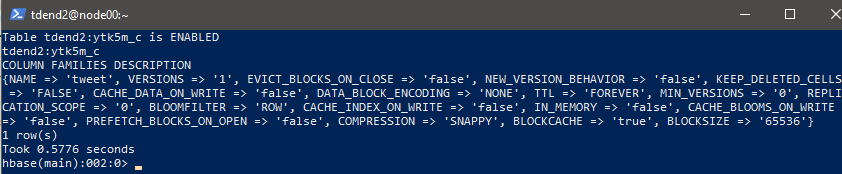
- -

* Check and show the tables’ descriptions and highlight (or mark) the compress

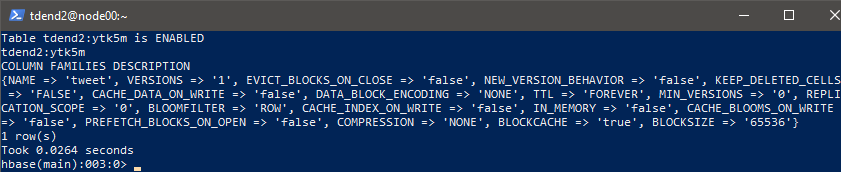
Configuration.

# Execution:

describe 'tdend2: ytk5m\_c'



describe 'tdend2: ytk5m'



* Load the data (ytk5m.csv) into your two tables in HBase.

# Execution:

Step-1: Exit hbase shell

Step-2: Loaded csv file into tables as follows:

Usage: importtsv -Dimporttsv.columns=a,b,c <tablename> <inputdir> Execution:

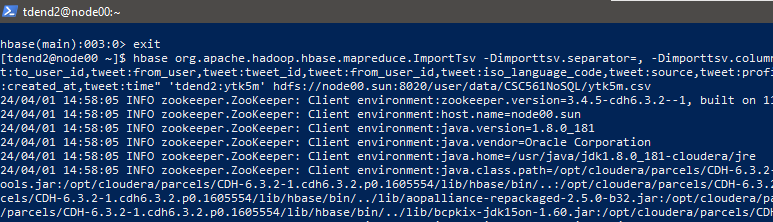
1. Loading csv to uncompressed table

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv - Dimporttsv.separator= , - Dimporttsv.columns="HBASE\_ROW\_KEY,tweet:colnum,tweet:archivesource,tweet:text,tw eet:cleantext,tweet:to\_user\_id,tweet:from\_user,tweet:tweet\_id,tweet:from\_user\_id,tweet: iso\_language\_code,tweet:source,tweet:profile\_image\_url,tweet:geo\_type,tweet:geo\_coor dinates\_0,tweet:geo\_coordinates\_1,tweet:created\_at,tweet:time" 'tdend2:ytk5m' hdfs://node00.sun:8020/user/data/CSC561NoSǪL/ytk5m.csv

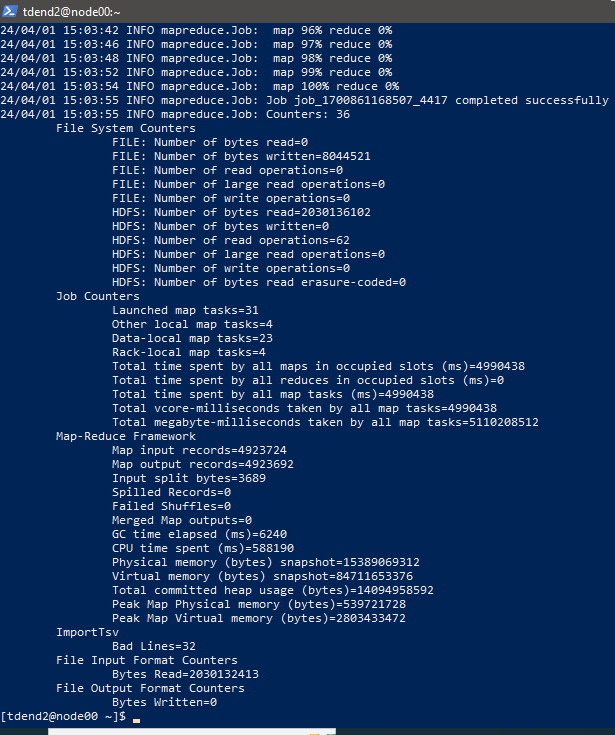
1. b) Loading csv to compressed table

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv - Dimporttsv.separator= , - Dimporttsv.columns="HBASE\_ROW\_KEY,tweet:colnum,tweet:archivesource,tweet:text,tw eet:cleantext,tweet:to\_user\_id,tweet:from\_user,tweet:tweet\_id,tweet:from\_user\_id,tweet: iso\_language\_code,tweet:source,tweet:profile\_image\_url,tweet:geo\_type,tweet:geo\_coor dinates\_0,tweet:geo\_coordinates\_1,tweet:created\_at,tweet:time" 'tdend2:ytk5m\_c' hdfs://node00.sun:8020/user/data/CSC561NoSǪL/ytk5m.csv

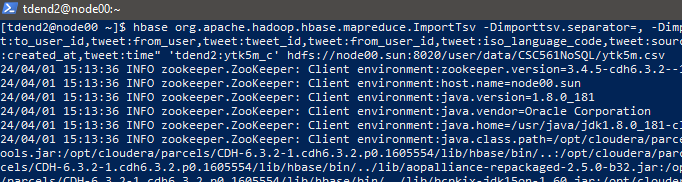
1. Result of loading csv to ‘tdend2:ytk5m table’:



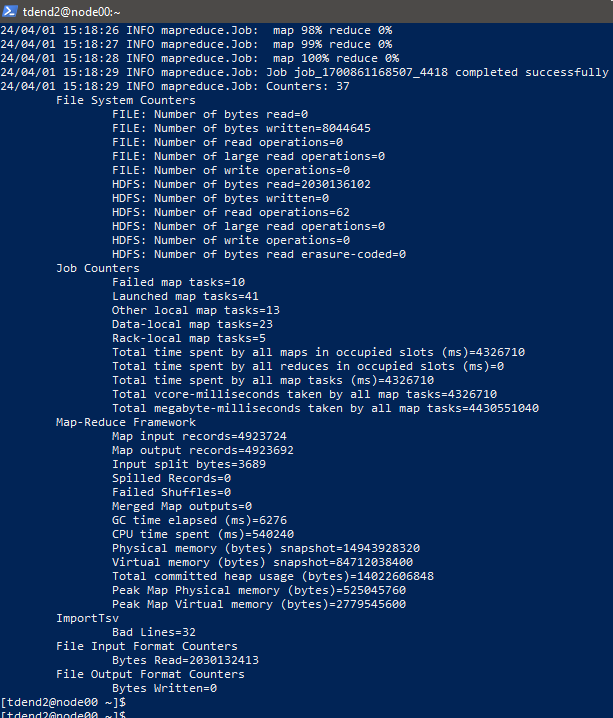
Last part of result:



1. Result of loading csv to compressed table:



Last half of result:



* Noticed that 32 lines of data are not loaded.
* Count the number of rows of the two tables to check that the loadings were successful.

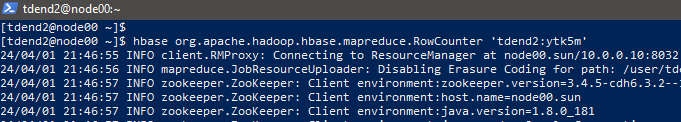
(Note: Double-check the number of rows. It should be around 5 million rows)

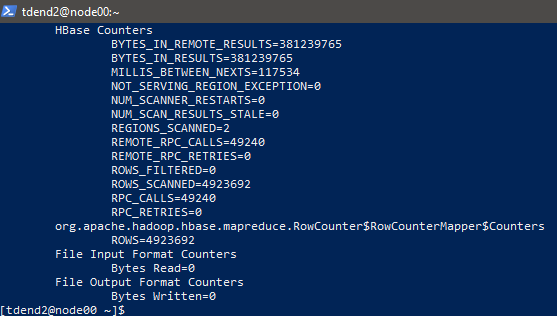
# Two different approaches to check the speed:

* 1) Use the ‘RowCounter’ class of HBase outside HBase (in Linux shell) and show

row counts and elapsed time.

**hbase org.apache.hadoop.hbase.mapreduce.RowCounter ‘tdend2:ytk5m’**

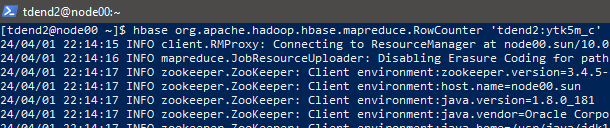


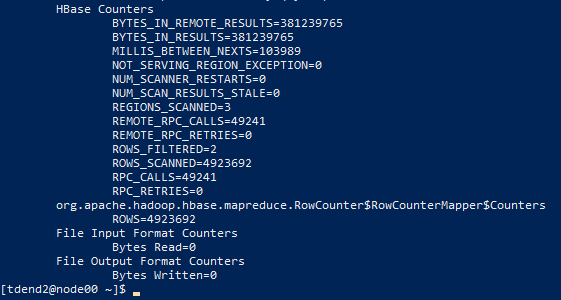


***Therefore, the start time from above is 21:46:55 and the timer stopped after 1:35:52 seconds. It took 99 seconds.***

# For compressed table ‘ytk5m\_c’:

Result:





## The timer started at 10:14:15 and ended at 10:15:43.

Calculate the difference:

**Elapsed time** for the execution of row count on compressed table using row counters: 88

# seconds.

## Hence, in this approach, compressed table took less time to complete the execution by 11 seconds.

1. Use the ‘count’ HBase function outside HBase and show row counts and elapsed time.

***Approach 2 – Count the rows using* count function of hbase**

# for uncompressed table ‘tdend2:ytk5m’:

Script:

# the HBase table name table\_name="tdend2:ytk5m\_c"

# the HBase shell command with INTERVAL option to reduce logs hbase\_command="count '$table\_name', INTERVAL => 1000000"

# Recording the start time start\_time=$(date +%s)

# Execute the HBase shell command and retrieve the output hbase\_output=$(echo "$hbase\_command" | hbase shell 2>&1)

# Recording the end time end\_time=$(date +%s)

# Calculation of the elapsed time elapsed\_time=$((end\_time - start\_time))

# Output the full HBase shell command output and elapsed time echo "HBase shell command output:"

echo "$hbase\_output"

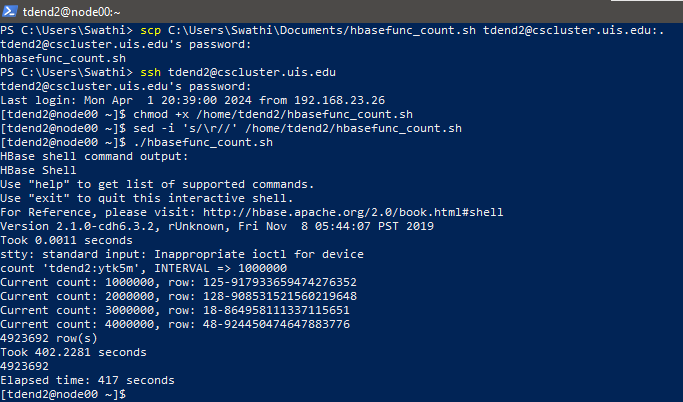
echo "Elapsed time: $elapsed\_time seconds"

# Steps of Implementation:

Step-1: Write the required code in the shell script Step-2: Copy shell script to cs cluster

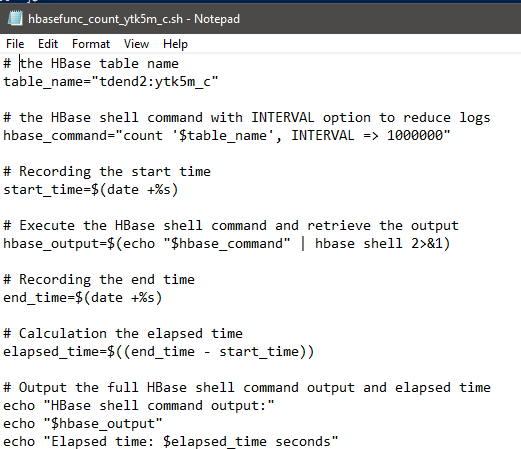
Step-3: Give required permissions to execute Step-4: Look for carriage returns (Optional)

Step-5: Execute the script using appropriate path to the file and command Result:



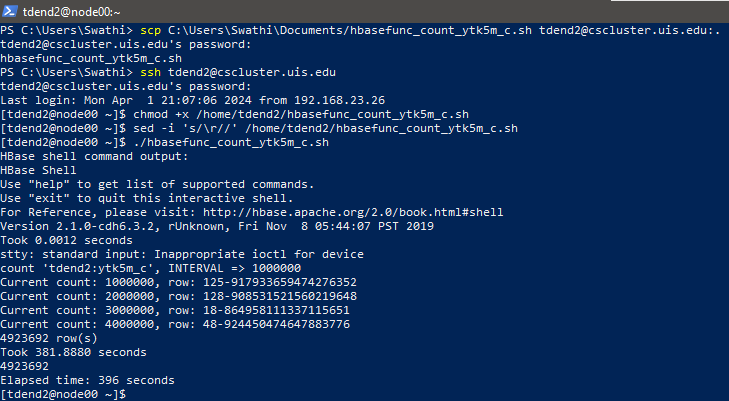
## Hence, the row count of uncompressed table is 4923692 and elapsed time is 417 seconds.

* 1. Compressed table: Script:



Here 2>&1 gives the error and output both to standard output and to the same display which is user friendly and convenient to know errors

Result of count function of counting rows in compressed table:



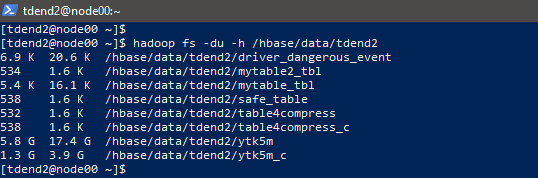
## Hence, the row count of compressed table is 4923692 and elapsed time is 396 seconds. So the compressed table took shorter duration to count the rows when compared to the uncompressed one and number of loaded from csv are same in both tables.

***Overall summary:***

## Comparing the above two approaches, performance of ‘Row counter ‘ is faster than the count function of hbase. In both the approaches compressed table performed faster than the uncompressed one.

* Check the file size of the tables in the Linux shell. Explain the differences. Execution:

hadoop fs -du -h /hbase/data/tdend2 Result:



The directory /hbase/data/tdend2/ytk5m occupies 5.8 gigabytes of disk space, and when considering its subdirectories, it occupies a total of 17.4 gigabytes. Similarly,

/hbase/data/tdend2/ytk5m\_c occupies 1.3 gigabytes of disk space, and a total of 3.9 gigabytes when considering its subdirectories.

***Hence, the compressed table ’ytk5m\_c’ occupied less space on disk compared to the uncompressed one and smaller in size by 13.5 gigabytes and occupied 4.5G less space on disk than the uncompressed one.***

* Created two external tables (filename: ytk5m and ytk5m\_c) in Hive by linking(mapping) the two tables in HBase.

# Execution:

**Note: used hive table names as hive\_table\_ytk5m, hive\_table\_ytk5m\_c**

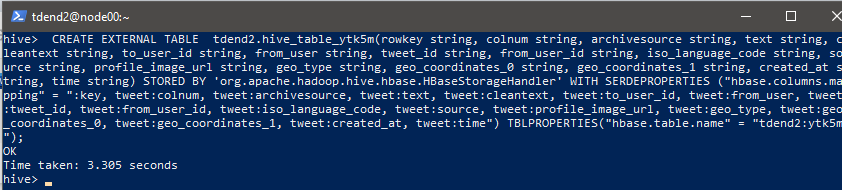
Step-1: Start hive to initialize hive shell

Step-2: Execution of mapping HBase table to hive table

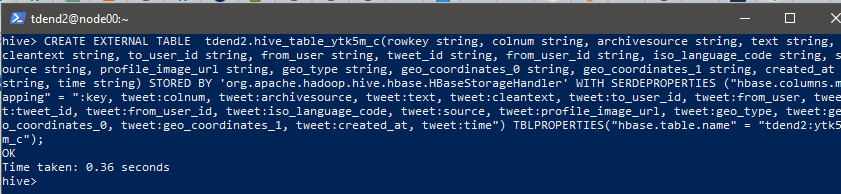
CREATE EXTERNAL TABLE tdend2.hive\_table\_ytk5m(rowkey string, colnum string, archivesource string, text string, cleantext string, to\_user\_id string, from\_user string, tweet\_id string, from\_user\_id string, iso\_language\_code string, source string, profile\_image\_url string, geo\_type string, geo\_coordinates\_0 string, geo\_coordinates\_1 string, created\_at string, time string) STORED BY

'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key, tweet:colnum, tweet:archivesource, tweet:text, tweet:cleantext, tweet:to\_user\_id, tweet:from\_user, tweet:tweet\_id, tweet:from\_user\_id, tweet:iso\_language\_code, tweet:source, tweet:profile\_image\_url, tweet:geo\_type, tweet:geo\_coordinates\_0, tweet:geo\_coordinates\_1, tweet:created\_at, tweet:time") TBLPROPERTIES("hbase.table.name" = "tdend2:ytk5m");

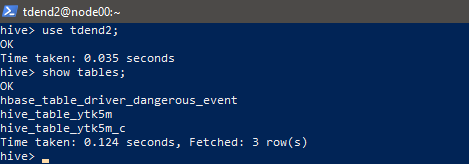
Result:



CREATE EXTERNAL TABLE tdend2.hive\_table\_ytk5m\_c(rowkey string, colnum string, archivesource string, text string, cleantext string, to\_user\_id string, from\_user string, tweet\_id string, from\_user\_id string, iso\_language\_code string, source string, profile\_image\_url string, geo\_type string, geo\_coordinates\_0 string, geo\_coordinates\_1 string, created\_at string, time string) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key, tweet:colnum, tweet:archivesource, tweet:text, tweet:cleantext, tweet:to\_user\_id, tweet:from\_user, tweet:tweet\_id, tweet:from\_user\_id, tweet:iso\_language\_code, tweet:source, tweet:profile\_image\_url, tweet:geo\_type, tweet:geo\_coordinates\_0, tweet:geo\_coordinates\_1, tweet:created\_at, tweet:time") TBLPROPERTIES("hbase.table.name" = "tdend2:ytk5m\_c");



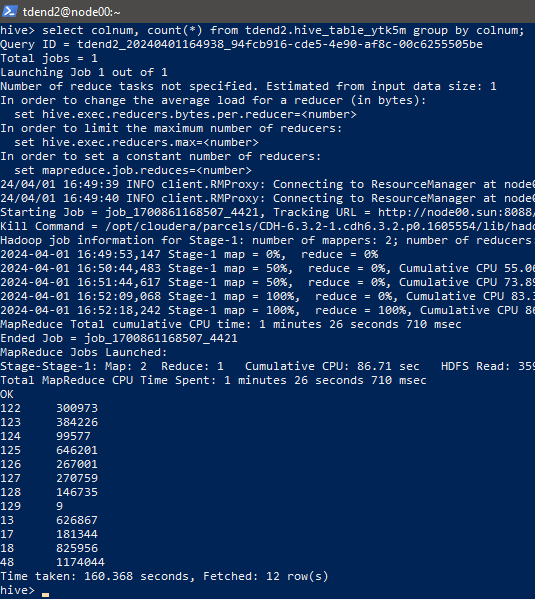
Listed tables:



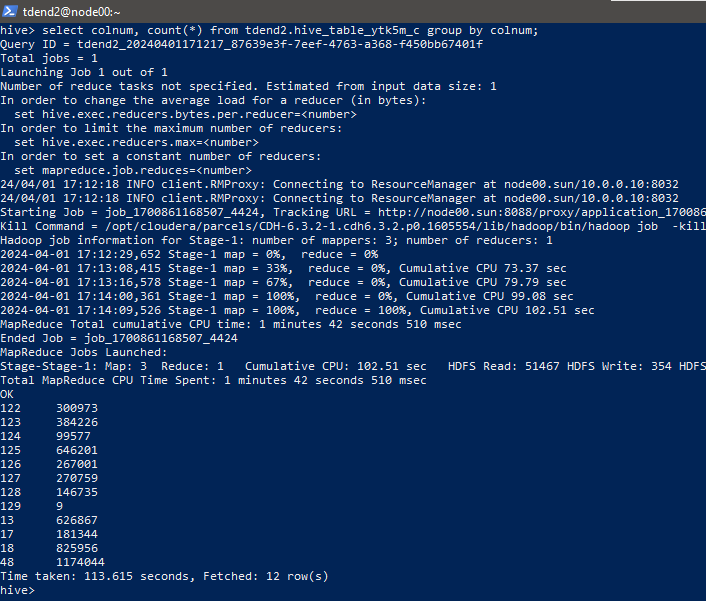
* Evaluate performances of the evaluation query against the tables.
* Used an evaluation query for both the tables correspondingly as below:

select colnum, count(\*) from tdend2.hive\_table\_ytk5m group by colnum; select colnum, count(\*) from tdend2.hive\_table\_ytk5m\_c group by colnum;

Execution:



Execution of evaluation query for compressed table:

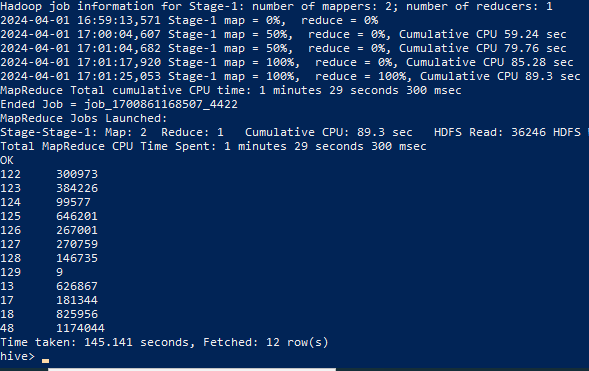


# Evaluation of performance:

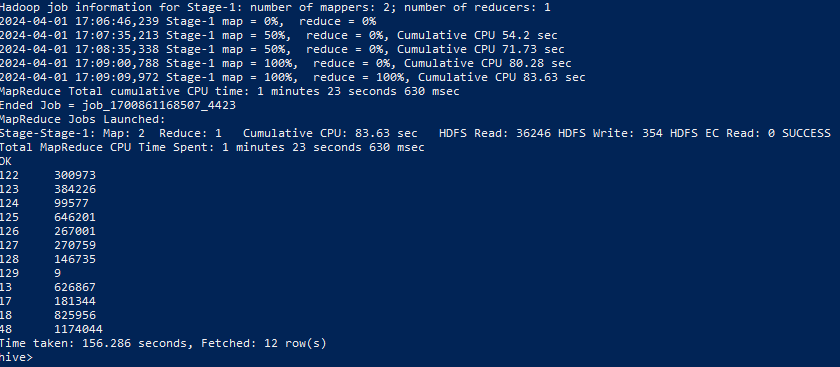
* To evaluate the performance of both compressed and uncompressed hive tables, calculated average elapsed time by running the above queries 3 times.

**Time taken** to execute the above evaluation queries both tables is as follows: a)Uncompressed table:

First execution: 160.368 seconds



Second execution: 145.141 seconds



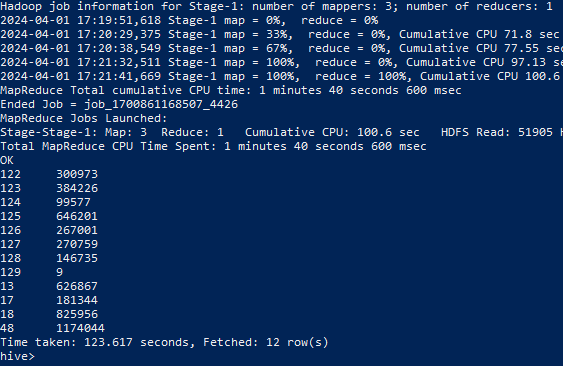
Third execution: 156.286 seconds

## Average elapsed time of uncompressed hive table:

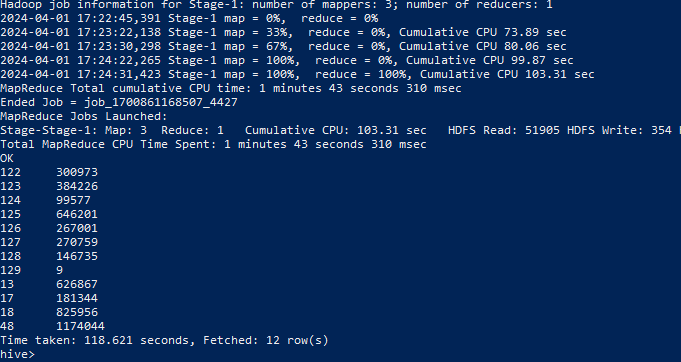
(156.286+160.368+145.141)/3= 153.931666667 ~**153.932 seconds**

a)Compressed table:

First execution: 113.615seconds



Second execution: 123.617 seconds



Third execution: *118.621*

## Calculation of Average elapsed time of compressed hive table:

*113.615+123.617+118.621= 118.617666667~****118.618 seconds.***

## Hence, it can be concluded that performance of compressed table is faster than the uncompressed one by 35.314 seconds as average elapsed time of compressed hive table is 118.618 seconds which is smaller than the uncompressed one’s which is as 153.932 seconds.

===============================THE END=================================