# CA675: Cloud Technologies Assignment 1: Stack Exchange Data Analysis in Hadoop

Dao Thi Nguyen
*StudentId: 20212316*
*DC836.MSc Computing.Data Analytics*
Email: dao.nguyen7@mail.dcu.ie

***Abstract*—This report illustrates a Big Data analysis project using Cloud Technologies. The data is retrieved from StackExchange and processed in Hadoop. In particular, the data is cleaned using Pig before being inserted into Hive for querying and analysis.**

***Index Terms*—CA675, Pig, Hive, Hadoop**

## I. Introduction

This project aims to process and analyse big data using Cloud Technologies. The data is extracted from StackExchange, including data about question-and-answer on topics in various fields. The objectives are to solve four following tasks:

1) Getting data from Stack Exchange - The solution is presented in Section II
2) Loading the data with PIG - The solution is presented in Section III
3) Querying data with Hive - The solution is presented in Section IV
4) Calculating TF-IDF with MapReduce/Pig/Hive - The solution is presented in Section V

The source codes and description detail of this project can also be found on GitHub at the folowing url [1]:

https://github.com/dao-nguyen0912/CA675_cloud-technologies_assignment1

## II. TASK 1 - Data acquisition

The task is requiring the data including top 200,000 posts by ViewCount from Stack Exchange [2]. However, the website is allowed only 50.000 records can be downloaded at a time. Therefore, it needs five queries to collect all necessary datasets for this project.

First, the threshold of the number of ViewCount of the top 200.000 posts needs to be identified. Using a single query to order all rows would take a long execution time. Instead, it is much faster by querying posts whose ViewCount is greater than a particular value. After a few tests, it is turned out that there are 200.555 posts having more than 36,500 views. The query can be seen below:

```
SELECT count(*) FROM Posts
WHERE ViewCount > 36500;
```

The next steps is using this ViewCount threshold to query and collect data from StackExchange, which can be seen in the following five queries. The first query returns the top 50,000 posts whose ViewCount is greater than 36,500. The lowest ViewCounts in the returned results of the first query (i.e. 111,930) can be used to narrow down the condition in the second query. Similarly, the lowest ViewCounts in the result of the second query (65,887) can be used to determine the condition for the third query, and so on.

```
SELECT TOP 50000 * FROM Posts
WHERE ViewCount > 36500
ORDER BY ViewCount DESC;

SELECT TOP 50000 * FROM Posts
WHERE ViewCount > 36500
AND ViewCount < 111930
ORDER BY ViewCount DESC;

SELECT TOP 50000 * FROM Posts
WHERE ViewCount > 36500
AND ViewCount <= 65887
ORDER BY ViewCount DESC;

SELECT TOP 50000 * FROM Posts
WHERE ViewCount > 36500
AND ViewCount <= 47040
ORDER BY ViewCount DESC;

SELECT TOP 50* FROM Posts
WHERE ViewCount > 36500
AND ViewCount <= 36591
ORDER BY ViewCount DESC;
```

Although the first four queries provided 200.000 rows, more 50 posts has been included in case there was a possible duplication in the retrieved data, which is the purpose of the fifth query. The results of those queries above were downloaded as 5 csv files namely *QueryResults1.csv, QueryResults2.csv, QueryResults3.csv, QueryResults4.csv and QueryResults5.csv*, respectively, which can be used for task 2.

## III. TASK 2 - Load data with PIG

In this task, a Hadoop cluster namely *cluster-a7d8-m* on *Google Cloud Platform* has been used as a working environment to implement data cleaning and querying with *Pig* and *Hive*. The five CSV files obtained from TASK 1 were uploaded to the cluster and then put to *HDFS*. The result can be seen in Figure 1.

It is the fact that the entry data contains a field with line-break characters. Consequently, the functions such as CSVLoader and PigStorage did not work properly to handle that issue. Instead, CSVExcelStorage has been selected due to its support for loading multi line data. This function is

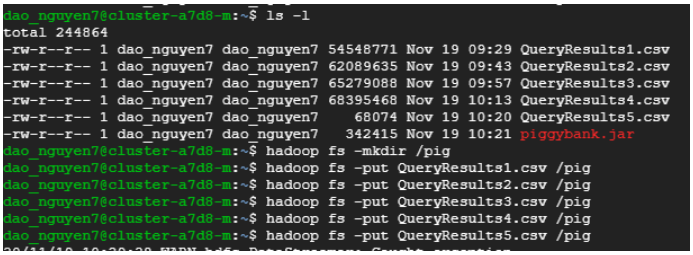


Fig. 1. Uploaded files in the cluster and HDFS

available in the *piggybank* library which can be downloaded at [3] and registered into Hadoop as follows:

```
grunt> register piggybank.jar
```

The following command is used to load data from the five CSV files into *Pig*. The character **X** in the following commands corresponds to the file name number, i.e. X is in the range of [1..5]. Please note that these five datasets, after being cleaned, will be merged into a single dataset for analysis in *Hive* in TASK 3.

```
loadDataX = Load 'hdfs://cluster-a7d8-m/pig/
    ↪ QueryResultsX.csv' USING org.apache.pig.
    ↪ piggybank.storage.CSVExcelStorage(',','
    ↪ YES_MULTILINE','WINDOWS','SKIP_INPUT_HEADER'
    ↪ ) AS (Id:int,PostTypeId:int,AcceptedAnswerId
    ↪ :int,ParentId:int,CreationDate:datetime,
    ↪ DeletionDate:datetime,Score:int,ViewCount:
    ↪ int,Body:chararray,OwnerUserId:int,
    ↪ OwnerDisplayName:chararray,LastEditorUserId:
    ↪ int,LastEditorDisplayName:chararray,
    ↪ LastEditDate:datetime,LastActivityDate:
    ↪ datetime,Title:chararray,Tags:chararray,
    ↪ AnswerCount:int,CommentCount:int,
    ↪ FavoriteCount:int,ClosedDate:datetime,
    ↪ CommunityOwnedDate:datetime,ContentLicense:
    ↪ chararray);
```

In the next step, the loaded data can be generated into a table with only necessary fields. Specially, the field *Body* has been cleaned using *REPLACE* functions. In particular, line-break characters (i.e. \n, \r) and HTML tags has been eliminated and replaced by space characters. It is an essential process so that the data can be properly loaded into *Hive* tables in the later tasks.

```
grunt> generateDataX = FOREACH loadDataX GENERATE Id
    ↪ , Score, ViewCount, OwnerUserId,
    ↪ OwnerDisplayName, Title, Tags, REPLACE(
    ↪ REPLACE(REPLACE(Body,'[\n|\r]','␣'),',','␣'),
    ↪ '<[^>]*>','␣');
```

The generated data can be stored into HDFS /FinalHiveX (with X is in the range of [1:5])

```
STORE generateDataX INTO '/FinalHiveX' USING org.
    ↪ apache.pig.piggybank.storage.CSVExcelStorage(
    ↪ ',','YES_MULTILINE','WINDOWS','
    ↪ SKIP_INPUT_HEADER');
```

After the storage, *Pig* has divided the result in 2 files *SUCCESS* and *part-m-00000* in */FinalHiveX* in *HDFS*. The log file namely SUCCESS will block the load function of *Hive* so this file need to be deleted with the following command:

```
hadoop fs -rm /FinalHiveX/_SUCCESS
```

The result files can be renamed to CSV files and copied to the local machine on the cluster as follows:

```
hadoop fs -mv /FinalHiveX/part-m-00000 /HiveDataX.
    ↪ csv

hadoop fs -copyToLocal 'hdfs://cluster-a7d8-m/
    ↪ HiveDataX.csv' /home/dao_nguyen7
```

The result can be seen in Figure 2

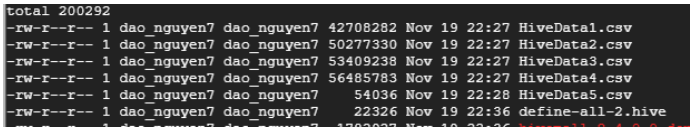


Fig. 2. Files in the local machine on the cluster

At this stage, the data has been cleaned and ready to be loaded and analysed with *Hive* in Task 3.

## IV. TASK 3 - QUERY DATA WITH HIVE

First, the generated data in CSV files extracted from Task 2 can be loaded into *Hive* as follows. Please note that the character **X** in the commands below correspond to the file name number from 1 to 5. As a result, there are five tables (*hiveTable2, hiveTable2* etc.) being created.

```
CREATE external TABLE IF not exists hiveTableX (Id
    ↪ int, Score int, ViewCount int, OwnerUserId
    ↪ int, OwnerDisplayName string, Title string,
    ↪ Tags string, Body string) ROW FORMAT
    ↪ DELIMITED FIELDS TERMINATED BY ',';

Load data local inpath 'HiveDataX.csv' overwrite
    ↪ into table hiveTableX;
```

The five tables created above can be merged into a single table *hiveTable* as follows. The result can be seen in Figure 3

```
INSERT OVERWRITE TABLE hiveTable SELECT * FROM
    ↪ hiveTable1 UNION ALL SELECT * from hiveTable2
    ↪  UNION ALL SELECT * from hiveTable3 UNION ALL
    ↪  SELECT * from hiveTable4 UNION ALL SELECT *
    ↪ from hiveTable5;
```

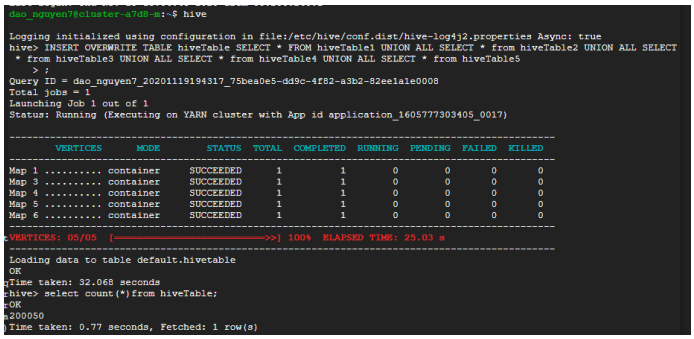


Fig. 3. The results of inserting all data in hiveTable with total of 200.050 rows

The table including top 200.000 posts by ViewCount after removing duplicated rows can be extracted as follows:

```
Create TABLE hiveTable_ok as SELECT distinct * FROM
    ↪ hiveTable order by viewcount desc limit
    ↪ 200000;
```

From the query above, the table *hiveTable-ok* is ready to use for the querying tasks in the following sub-sections.

### A. The top 10 posts by score

```
select id, title from hiveTable limit 10;
```

The result can be seen in Figure 4 which indicates the top 10 posts having the highest scores in the dataset.

```
hive> select id, title from hiveTable limit 10;
OK
927358  How do I undo the most recent local commits in Git?
2003505 How do I delete a Git branch locally and remotely?
5767325 How can I remove a specific item from an array?
16956810        How do I find all files containing specific text on Linux?
2906582 How to create an HTML button that acts like a link?
503093  How do I redirect to another webpage?
4114095 How do I revert a Git repository to a previous commit?
1789945 How to check whether a string contains a substring in JavaScript?
5585779 How do I convert a String to an int in Java?
1783405 How do I check out a remote Git branch?
Time taken: 0.236 seconds, Fetched: 10 row(s)
hive>
```

Fig. 4. The result of the top 10 post by score

### B. The top 10 users by post score

```
hive> select ownerUserid, sum(Score) as score_sum
    ↪ from hiveTable_ok group by ownerUserId order
    ↪ by score_sum desc limit 10;
```

The result can be seen in Figure 5

```
hive> select ownerUserid, sum(Score) as score_sum from hiveTable_ok group by ownerUserId order by score_sum desc li
mit 10;
Query ID = dao_nguyen7_20201119201401_4b082ff1-d293-46c6-a1cc-5d965bf3e311
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1605777303405_0020)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ..... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ..... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 23.22 s
--------------------------------------------------------------------------------
OK
NULL    407834
87234   36212
4883    26933
9951    25391
6068    24571
89904   22425
51816   21269
49153   18829
95592   18291
63051   18143
Time taken: 36.002 seconds, Fetched: 10 row(s)
hive>
```

Fig. 5. the result of top 10 users by score

The first ownerUserId is NULL as can be seen in Figure 5. This is because the values of this field are missing in many posts in the original datasets.

As one may notice that the result above merely provided *user Id*. It is also possible to query the *user displayed name* by the following queries. Please note that one *user Id* may have various *user displayed names* as can be seen in Figure 6.

```
create table user_top10 as select ownerUserid, sum(
    ↪ Score) as score_sum from hiveTable_ok group
    ↪ by ownerUserId order by score_sum desc limit
    ↪ 10;

hive> select ownerUserId, ownerDisplayName from
    ↪ hiveTable_ok where ownerUserId in (select
    ↪ ownerUserId from user_top10) group by
    ↪ ownerUserId, ownerDisplayName;
```

```
hive> select ownerUserId, ownerDisplayName from hiveTable_ok where ownerUserId in ( select ownerUserId from user_to
p10) group by ownerUserId, ownerDisplayName;
Query ID = dao_nguyen7_20201119203340_5569bddd-2ebc-43bb-9d03-362662d192a7
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1605777303405_0021)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ......... container     SUCCEEDED      4          4        0        0       0       0
Map 3 ......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ..... container     SUCCEEDED      1          1        0        0       0       0
Reducer 4 ..... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 04/04  [==========================>>] 100%  ELAPSED TIME: 22.33 s
--------------------------------------------------------------------------------
OK
4883
4883    Readonly
4883    Tim
6068
6068    J. Pablo Fern&#225;ndez
9951
9951    e-satis
49153
49153   Click Upvote
49153   Java
49153   Java PHP
51816
51816   Joan Venge
63051
63051   andy
87234
89904
95592
Time taken: 23.367 seconds, Fetched: 18 row(s)
```

Fig. 6. Top 10 users by score with userDisplayName

### C. The number of distinct users, who used the word "Hadoop" in one of their posts

The above task can be done via the hive command below. The result can be seen in Figure 7 as there are 134 distinct users using the word *"Hadoop"* in at least one of their posts.

```
hive> select COUNT(distinct(OwnerUserId)) from
    ↪ hiveTable_ok where Body like '%Hadoop%' or
    ↪ Title like '%Hadoop%' or Tags like '%Hadoop%'
    ↪ ;
```

```
hive> select COUNT(distinct(OwnerUserId)) from hiveTable_ok where Body like '%Hadoop%' or Title like '%Hadoop%' or
Tags like '%Hadoop%';
Query ID = dao_nguyen7_20201119204208_eada6903-f842-48cd-9b67-2f04f916ae54
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1605777303405_0022)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ..... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ..... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 18.96 s
--------------------------------------------------------------------------------
OK
134
Time taken: 30.479 seconds, Fetched: 1 row(s)
hive>
```

Fig. 7. Distinct users, who used the word 'Hadoop' in a post

## V. TASK 4 - CALCULATE TF-IDF

The task objective is to utilise *MapReduce/Pig/Hive* for calculating the per-user *TF-IDF*, reporting the top 10 terms for each of the top 10 users from the result in Task 3.B. In this task, Apache Hivemall will be utilised. It provides machine learning functionality as well as feature engineering functions through *UDFs/UDAFs/UDTFs* of *Hive*. It also offers *TF-IDF* tool [4] for retrieval information. The calculation of *TF-IDF* is described below:

1. Create a table with data retrieved from hiveTable and *user_top10* which includes top 10 highest score users.

```
create table question4Data as select ownerUserId,
    ↪ Title, Body, Tags from hiveTable where
    ↪ ownerUserId in (select ownerUserId from
    ↪ user_top10);
```

2. Upload and add jar and hive files in Hive to load necessary functions such as *tokenize()* and *is_stopword()*. These files can be download at [5].

```
hive> add jar hivemall-0.4.2.2-dependencies.jar
hive> source define-all-2.hive
```

3. Define macro used for the *TF-IDF* computation.

```
create temporary macro max2(x INT, y INT)
if(x>y,x,y);
create temporary macro tfidf(tf FLOAT, df_t INT,
    ↪ n_docs INT) tf * (log(10, CAST(n_docs as
    ↪ FLOAT)/max2(1,df_t)) + 1.0);
```

4. Create Views to display calculated *TF-IDF*;

```
create or replace view question4_exploded as select
    ↪ ownerUserId, word from question4Data LATERAL
    ↪ VIEW explode(tokenize(Body,true)) t as word
    ↪ where not is_stopword(word);


create or replace view term_frequency_temp as select
    ↪  ownerUserId, word, freq from (select
    ↪ ownerUserId, tf(word) as word2freq from
    ↪ question4_exploded group by ownerUserId) t
    ↪ LATERAL VIEW explode(word2freq) t2 as word,
    ↪ freq;
create or replace view term_frequency as select *
    ↪ from (select ownerUserId, word, freq, rank()
    ↪ over ( partition by ownerUserId order by freq
    ↪  desc) as rank from term_frequency_temp ) t
    ↪ where rank <= 10;


create or replace view document_frequency as select
    ↪ word, count(distinct ownerUserId) docs from
    ↪ question4_exploded group by word;
```

5. Set the total number of documents and display the result in a view;

```
set hivevar:n_docs=10;


create or replace view tfidf as select tf.
    ↪ ownerUserId, tf.word, tfidf(tf.freq, df.docs,
    ↪  ${n_docs}) as tfidf from term_frequency tf
    ↪ JOIN document_frequency df ON (tf.word = df.
    ↪ word) order by tfidf desc;
select ownerUserId, collect_list(feature(word, tfidf
    ↪ )) as features from tfidf group by
    ↪ ownerUserId;
```

The result can be seen in Figure 8 and 9

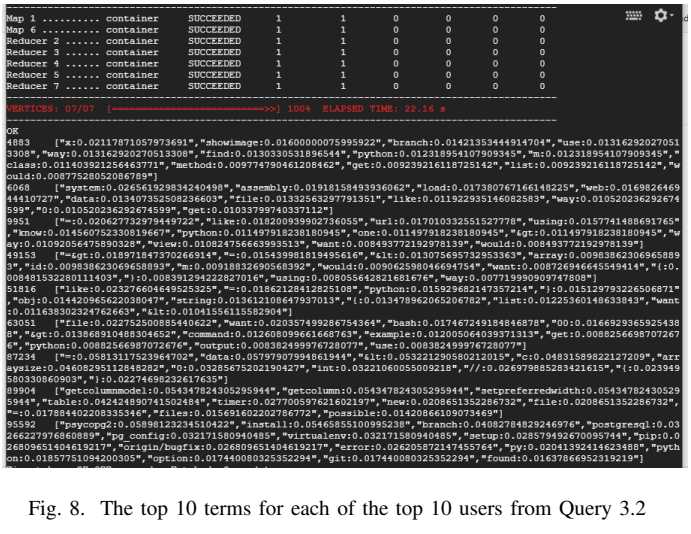


Fig. 8. The top 10 terms for each of the top 10 users from Query 3.2

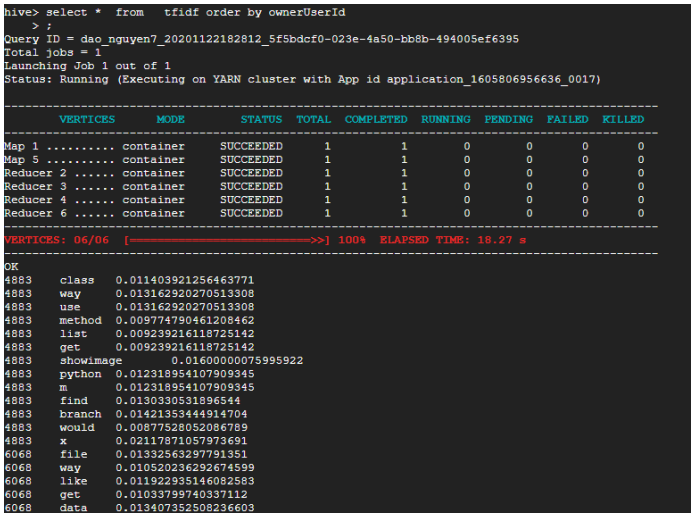


Fig. 9. The top 10 terms for each of the top 10 users from Query 3.2, top term list

It is noticed that there are some special characters, which are meaningless, contained in the result. One way to overcome this problem is cleaning these special characters by the function *REPLACE* in the Generate command in Task 2. This process requires a thorough research into patterns of the special characters to be removed without affecting the content, and, therefore, are out of the scope of this project.

## VI. Conclusion

This report has documented a Big Data analysis project in Hadoop environment. The raw data is retrieved from Stack-Exchange, which is relatively large and complicated. The data contain fields with special characters such as line-breaks and HTML tags. The data is loaded using the *piggybank* library with the function *CSVExcelStorage()* and cleaned by removing line-breaks and HTML tags in Pig. The pre-processed data is loaded into Hive for querying according to the specific requirements. The Hivemall is utilised to calculate TF-IDF of the terms in the data, which are displayed in corresponding Views.

## References


[1] "Dao-nguyen0912/CA675_cloud-technologies_assignment1," https://github.com/dao-nguyen0912/CA675_cloud-technologies_assignment1.

[2] "Query Stack Overflow - Stack Exchange Data Explorer," https://data.stackexchange.com/stackoverflow/query/new.

[3] "PiggyBank - Apache Pig - Apache Software Foundation," https://cwiki.apache.org/confluence/display/PIG/PiggyBank.

[4] "TF-IDF Term Weighting · Hivemall User Manual," https://hivemall.incubator.apache.org/userguide/ft_engineering/tfidf.html.

[5] "Installation · myui/hivemall Wiki," https://github.com/myui/hivemall/wiki/Installation.