LARGE DATASET ANALYSIS

Noha Alawwad

17214080

Submitted as a requirement for the course CA675P: Cloud Technologies.

MSc in Data Analytics. Spring 2018

Introduction

This assignment aims to apply many analysis tasks on a large data set. *Stack Exchange* website is the source of the data for this assignment. After data has been collected from *Stack Exchange* website by using SQL command (Task 1), the analysis of this data started by using Pig, Hive and MapReduce functions to perform three tasks: data loading with Pig, data querying with Hive and calculating Term Frequency - Inverse Document Frequency (TF_IDF) by MapReduce functions (Task 2-4). Finlay, Amazon Web Services (AWS) was used to run all these tasks on the cloud (Task 5).

1. Task 1: Acquiring data

In this task, the top 200,000 posts by views count were acquired from *Stack Exchange* by using SQL command.

2. Task 2: Loading Data with Pig

In this task, the data was loaded using Pig. The preprocessing step is important at this stage to prepare the data format to be in a proper way.

In the preprocessing step, many *sed* commands were used to keep the file clean. The first command was used to replace each new line with " /n", while the second one was used to remove the "/n" symbol which is resulted in removing the new lines as shown on Figure 2.1.

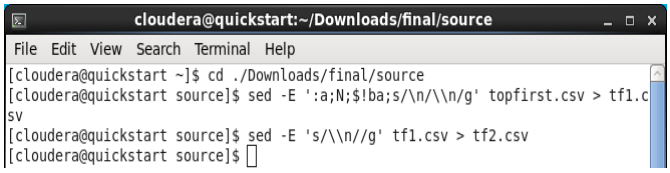


Figure 2.1: Sed command on the terminal

The *sed* command was used to remove punctuation and any special character except double quotation and comma to prevent changing the structure of the file. Figure 2.2 shows the list of *sed* command that has been applied on the data.

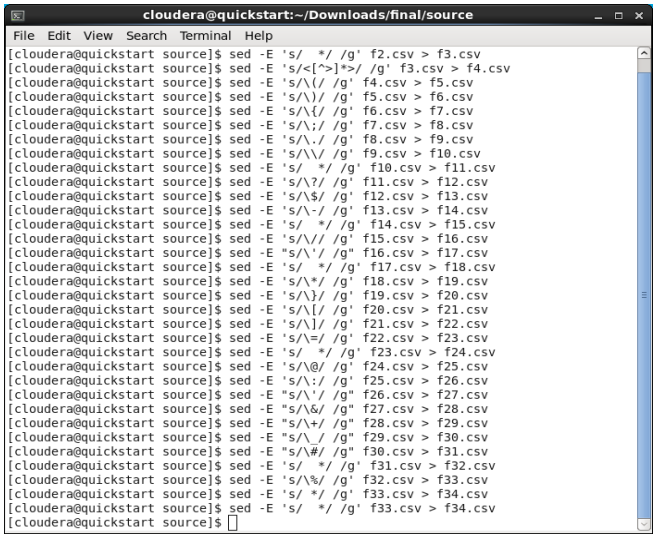


Figure 2.2: The list of sed command

After the preprocessing step, the file was ready to be on Hadoop by creating folder *final* and then assigning the five preprocessed files to the *final* folder location. Figures 2.3-a and 2.3-b show the steps of identifying the folder and the files on the Hadoop cluster.

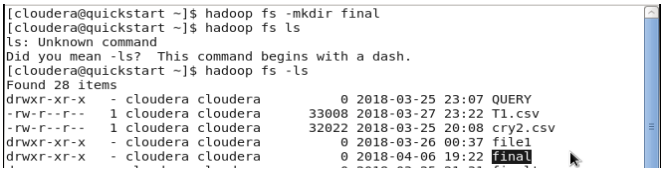


Figure 2.3-a: The identification of the folder on Hadoop cluster

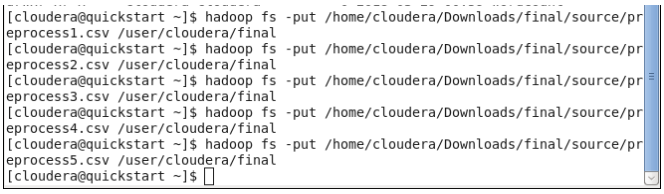


Figure 2.3-b: The identification of the files on Hadoop cluster

Next step, two tables were created with specific columns that are needed for queries using *Hive*. The first table has *Id*, *Score*, *Ownerdisplayname* and *Title* columns. The second table has *Owneruserid, Body* and *Title* columns. Figures 2.4-a and 2.4-b show the *Hive* command for creating two external tables.

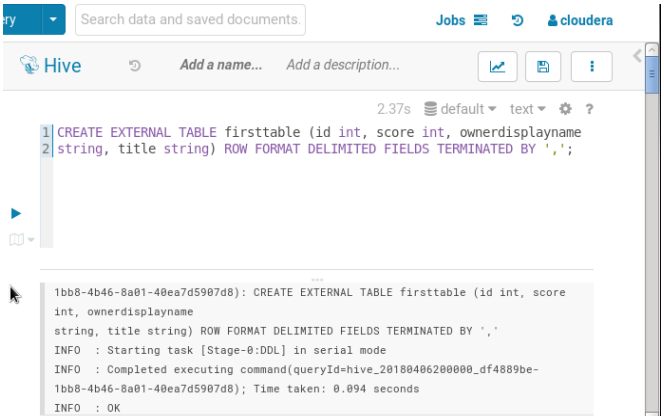


Figure 2.4-a: First table creation using hive command

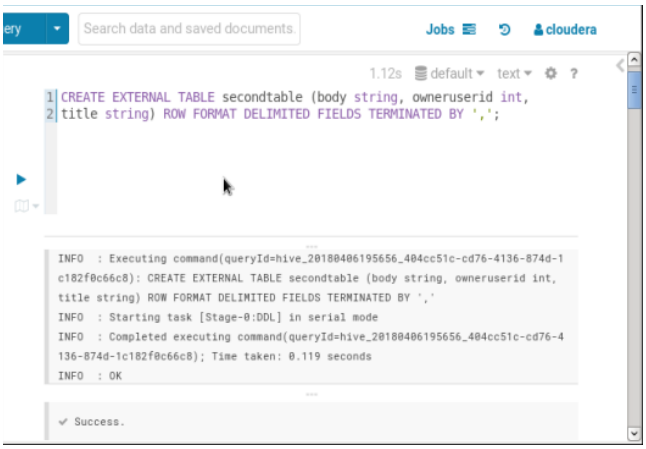


Figure 2.4-b: Second table creation using hive command

Then, *REGISTER* command was used to import *Piggybank*.jar library as shown on Figure 2.4-c

```
grunt> REGISTER /usr/lib/pig/piggybank.jar;
2018-04-06 20:14:19,153 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-04-06 20:14:19,153 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
grunt>
```

Figure 2.4-b: importing Piggybank.jar

In order to load the files with comma separated format, the *CSVExcelStorage* command from *org.apache.pig.piggybank.storage* was used. In this step, the header of each file was removed. Figures 2.5-a and 2.5-b show the loading step for each file using *CSVExcelStorage*.

Figure 2.5-a: Loading some files using CSVExcelStorage

```
grunt> topfifth = LOAD '/user/cloudera/final/preprocess5.csv' USING org.apache.p
ig.piggybank.storage.CSVExcelStorage(',', 'NO_MULTILINE', 'NOCHANGE','SKIP_INPUT
_HEADER') AS (Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, Creati
onDate:datetime, DeletionDate:datetime, Score:int, ViewCount:int, Body:chararray
, OwnerUserId:int, OwnerDisplayName:chararray, LastEditorUserId:int, LastEditorD
isplayName:chararray, LastEditDate:datetime, LastActivityDate:datetime, Title:ch
ararray, Tags:chararray, AnswerCount:int, CommentCount:int, FavoriteCount:int, C
losedDate:chararray, CommunityOwnedDate:datetime);
grunt>
```

Figure 2.5-b: Loading fifth file using CSVExcelStorage

After loading the files, the comma in *Body* and *Title* columns was replaced with space for each file. Then, the *union* command was used for *Top* variable as shown in Figure 2.6.

```
grunt> F = FOREACH topfirst GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId,
 CreationDate, DeletionDate, Score, ViewCount, REPLACE(Body,',',' '), OwnerUserI
d, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, Last
ActivityDate, REPLACE(Title,',',' '), Tags, AnswerCount, CommentCount, FavoriteC
ount, ClosedDate, CommunityOwnedDate;
grunt> S = FOREACH topsecond GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewCoun
t, REPLACE(Body,',',' '), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivityD
ate, REPLACE(Title,',',' '), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> T = FOREACH topthird GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewCount
, REPLACE(Body,',',' '), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivityDa
te, REPLACE(Title,',',' '), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> FO = FOREACH topfourth GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewCou
nt, REPLACE(Body,',',' '), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivity
Date, REPLACE(Title,',',' '), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> FI = FOREACH topfifth GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewCoun
t, REPLACE(Body,',',' '), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivityD
ate, REPLACE(Title,',',' '), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> TOP = UNION F, S, T, FO, FI;
grunt>
```

Figure 2.6: The replace steps of comma in each file in pig shell.

In order to prepare the data for query and get readable result, only *Id, Body* and *Title Score* columns were used for these queries. Then, the double quotation for *Title* and *Body* columns was replaced with space as shown in Figure 2.7-a.

```
grunt> FILE1 = FOREACH TOP GENERATE Id AS id, Score AS score, OwnerDisplayName AS ownerdisplayname, $15 AS title;
grunt> FILE2 = FOREACH TOP GENERATE REPLACE($8,'"',' ') AS body, OwnerUserId AS owneruserid, REPLACE($15,'"',' ') AS title;
grunt>
```

Figure 2.7-a: The generating steps of some columns in pig shell

Here in Figure 2.7-b, files were stored using *HCatStorer()* to store them directly to Hive tables.

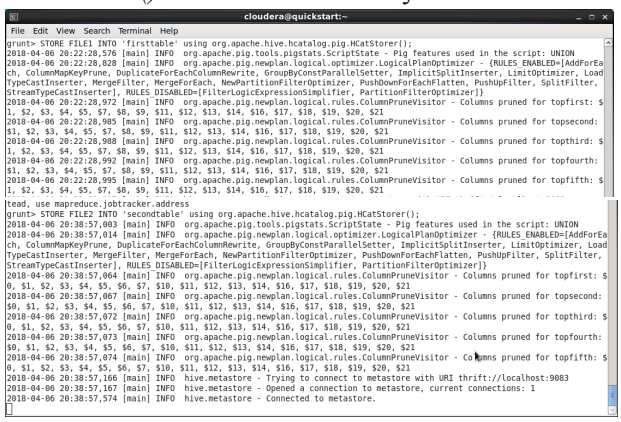


Figure 2.7-b: The stored process using HCatStorer() command

3. Task 3: Querying Data with Hive

Three queries have been answered as written below, and their output is shown in Figures 3.1, 3.2 and 3.3.

1. Get top 10 posts by score

```
select id, score, title from firsttable
sort by score desc
limit 10;
```

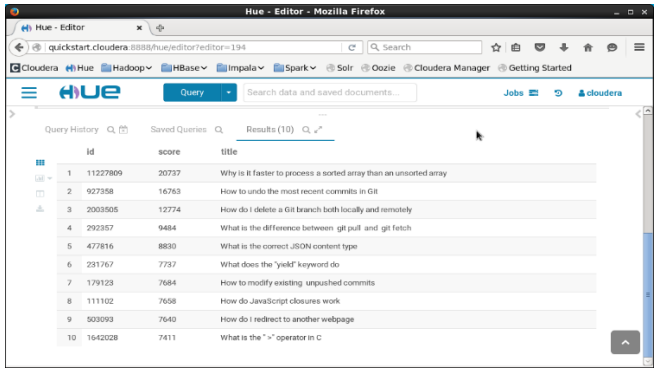


Figure 3.1: The output of the first query

2. Get top 10 users by post score

```
select id, score, ownerdisplayname  from firsttable
sort by score desc
limit 10;
```

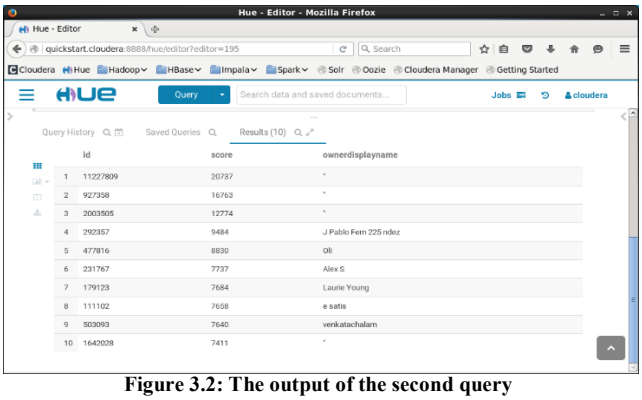


Figure 3.2: The output of the second query

3. Get the number of distinct users who used the word "hadoop" in one of their posts.

Select count (**distinct** (owneruserid) **from** secondtable **where** (body **REGEXP** 'hadoop') **or** (title **REGEXP** 'hadoop');

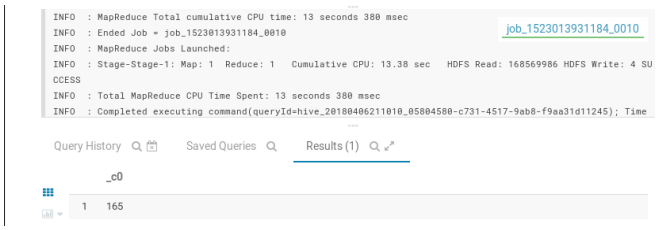


Figure 3.3: The output of the third query

4. Task 4: TF-IDF Algorithm

The TF-IDF was computed by MapReduce functions using python *mrjob* library. The most important columns in the dataset for this purpose is *user_owner_id*, *body and title.* These columns were loaded using *PigStorage* with comma to separate the columns as shown in Figure 4.1

```
grunt> A1 = load '/user/hive/warehouse/secondtable/part-m-00000' USING PigStorag
e(',') AS (Body:chararray, OwnerUserId:int, Title:chararray);
A2 = load '/user/hive/warehouse/secondtable/part-m-00001' USING PigStorage(',')
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A3 = load '/user/hive/warehouse/secondtable/part-m-00002' USING PigStorage(',')
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A4 = load '/user/hive/warehouse/secondtable/part-m-00003' USING PigStorage(',')
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A5 = load '/user/hive/warehouse/secondtable/part-m-00004' USING PigStorage(',')
AS (Body:chararray, OwnerUserId:int, Title:chararray);
grunt>
```

Figure 4.1: Loading columns using Pig storage

Then, the *body* and *title* columns were combined by using *CONCAT* command into one column called *text as* shown in Figure 4.2

```
AS (Body:chararray, OwnerUserId:int, Title:chararray);
grunt> AA1 = FOREACH A1 GENERATE $1, CONCAT($0,' ',$2) As text;
grunt> AA2 = FOREACH A2 GENERATE $1, CONCAT($0,' ',$2) As text;
grunt> AA3 = FOREACH A3 GENERATE $1, CONCAT($0,' ',$2) As text;
grunt> AA4 = FOREACH A4 GENERATE $1, CONCAT($0,' ',$2) As text;
grunt> AA5 = FOREACH A5 GENERATE $1, CONCAT($0,' ',$2) As text;
grunt>
```

Figure 4.2: Combining the body and title columns into text columns

Then *GENERATE* command was used as shown in Figure 4.3 to obtain two columns: *id* and *text*.

```
grunt>
grunt> AAA1 = FOREACH AA1 GENERATE $0,$1;
grunt> AAA2 = FOREACH AA2 GENERATE $0,$1;
grunt> AAA3 = FOREACH AA3 GENERATE $0,$1;
grunt> AAA4 = FOREACH AA4 GENERATE $0,$1;
grunt> AAA5 = FOREACH AA5 GENERATE $0,$1;
grunt>
```

Figure 4.3: Obtaining id and text columns using *Generate* command

In addition, *Replace* command was used in Figure 4.4 to replace any number that falls in the *text* columns with space.

```
grunt> c1 = FOREACH AAA1 GENERATE $0, REPLACE($1,'[0-9]*','');
c2 = FOREACH AAA2 GENERATE $0, REPLACE($1,'[0-9]*','');
c3 = FOREACH AAA3 GENERATE $0, REPLACE($1,'[0-9]*','');
c4 = FOREACH AAA4 GENERATE $0, REPLACE($1,'[0-9]*','');
c5 = FOREACH AAA5 GENERATE $0, REPLACE($1,'[0-9]*','');
grunt>
```

Figure 4.4: Replacing numbers in the text column using *Replace* command.

The last step was to merge the files with *Union* command and store them using *Store* command as in Figure 4.5.

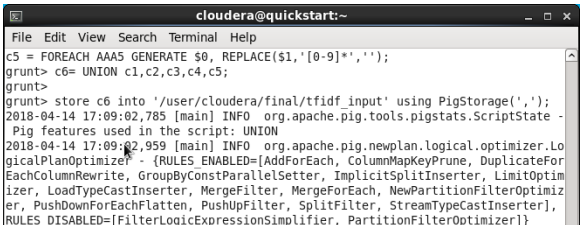


Figure 4.5: Union command to merge the files & Store command to store them

In order to use the resulted files from the previous step for the TF-IDF process, *cat* command was used to combine all the files as *input.csv* file as seen in Figure 4.6.

Figure 4.6: cat command to combine the files as input.csv file

The code of TF_IDF calculates the total number of distinct *user_owner_id* values and consider it as total number of documents. These distinct *user_owner_id* values are also used to represent documents' names, while each single *body* for the same *user_owner_id* needs to be gathered to represent one single document. The result of TF_IDF algorithm was saved in *Output.csv* file as shown in Figure 4.7.

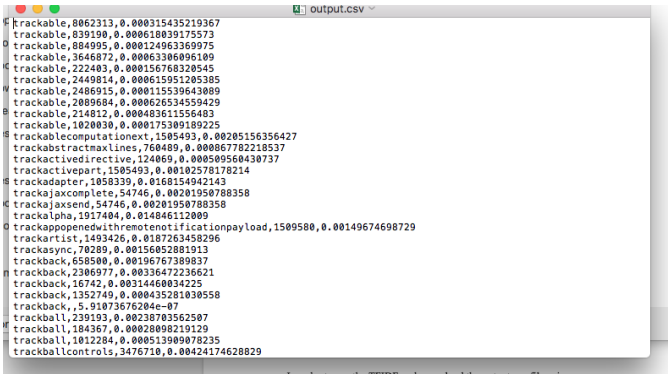


Figure 4.7: The output of TF_IDF algorithm

In order to calculate the top ten terms for each user, the *output.csv* file was moved to Hadoop cluster as in Figure 4.8.

```
[cloudera@quickstart ~]$ cd ./Downloads
[cloudera@quickstart Downloads]$ hadoop fs -put /home/cloudera/Downloads/output.
csv /user/cloudera/final
[cloudera@quickstart Downloads]$
```

Figure 4.8: moving the output file to Hadoop cluster

Then, Hive table was created as in Figure 4.9.

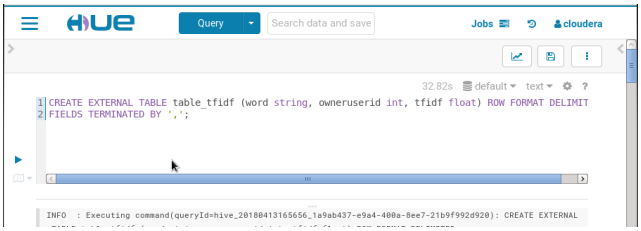


Figure 4.9: Hive table for TF-IDF

The output of the TF_IDF code, is loaded using *CSVExcelStorage command* containing *word, owner_user_id* and *tf_idf* columns then the table stored to Hive table using *Store* command as in Figure 4.10.

```
grunt> REGISTER /usr/lib/pig/piggybank.jar;
2018-04-13 17:10:14,722 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-04-13 17:10:14,723 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addr
ess
grunt> r = LOAD '/user/cloudera/final/output.csv' USING org.apache.pig.piggybank
.storage.CSVExcelStorage(',') AS (word:chararray, owneruserid:int, tfidf:float);

grunt> STORE r INTO 'table_tfidf' using org.apache.hive.hcatalog.pig.HCatStorer(
);
```

Figure 4.10: Loading output.csv file and storing it in Hive table

Then, the top 10 terms for each user were returned for specified user by using *Hive* query as shown in Figure 4.11. The output of the *Hive* query is shown in Figure 4.12.

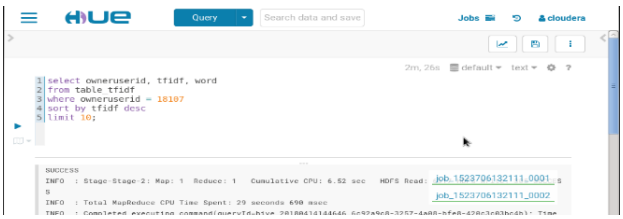
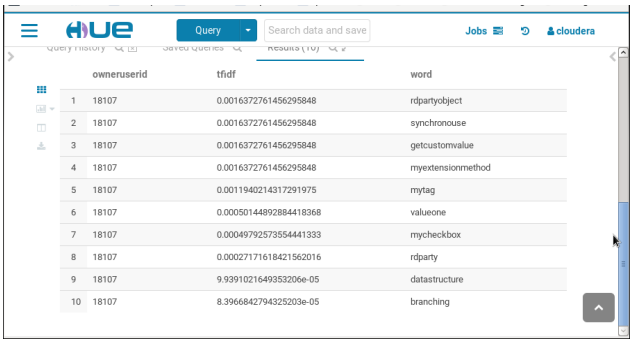


Figure 4.11: Hive query to return the top 10 terms



	owneruserid	tfidf	word
1	18107	0.0016372761456295848	rdpartyobject
2	18107	0.0016372761456295848	synchronouse
3	18107	0.0016372761456295848	getcustomvalue
4	18107	0.0016372761456295848	myextensionmethod
5	18107	0.0011940214317291975	mytag
6	18107	0.0005014489288441836 8	valueone
7	18107	0.0004979257355444133 3	mycheckbox
8	18107	0.0002717161842156201 6	rdparty
9	18107	9.939102164935320 6e-05	datastructure
10	18107	8.396684279432520 3e-05	branching

Figure 4.12: Output of the top 10 terms query

5. Task 5: Amazon Web Services (AWS)

At first, S3 bucket has been created with many folders, *data* for the input, *logs* to store the cluster logs, *results* to store the output files, and *scripts* folder for Pig and Hive scripts as Figure 5.1 shows.

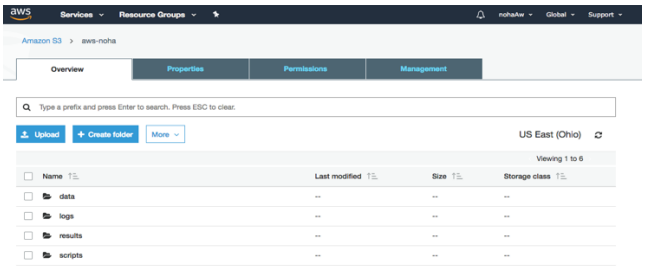


Figure 5.1: Create folders

Then, EC2 was used to create key pairs which used to connect AWS cloud with local computer to run the instance. Moreover, EMR was used to create cluster and steps for each task. Tasks applied in AWS as follows:

5.1. Loading data with Pig

Pig script assigned to the input data through creating Pig Program step in the cluster. Figure 5.2 shows the log of first pig script.

```
Input(s):
Successfully read 53222 records from: "s3://aws-noha/data/preprocess3.csv"
Successfully read 54995 records from: "s3://aws-noha/data/preprocess2.csv"
Successfully read 53582 records from: "s3://aws-noha/data/preprocess4.csv"
Successfully read 11 records from: "s3://aws-noha/data/preprocess5.csv"
Successfully read 54096 records from: "s3://aws-noha/data/preprocess1.csv"

Output(s):
Successfully stored 215906 records (12639302 bytes) in: "s3://aws-noha/results/res1"
Successfully stored 215906 records (168454096 bytes) in: "s3://aws-noha/results/res2"
```

Figure 5.2: pig script log

5.2. Querying data with Hive

Hive scripts were then launched as a hive program to create two tables, one for the first and second query and the other for the third query. Furthermore, Hive scripts for loading the data was created as new hive programs to load the resulted files from Pig program in order to query the data. Finally, three Hive programs were

created for the three needed queries. Figure 5.3 shows the cluster screen with the completed steps while the queries output is shown in Figure 5.4, 5.5, and 5.6.

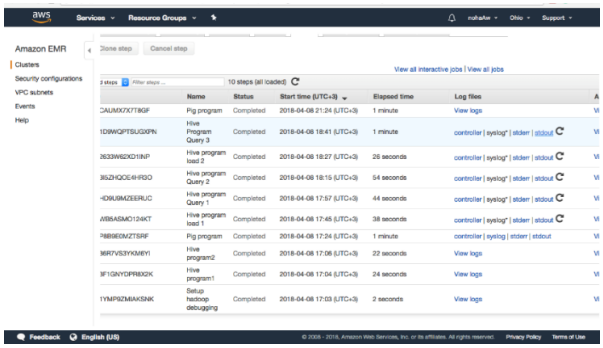


Figure 5.3: Completed steps

```
/usr/bin/hive
11227809        20737   Why is it faster to process a sorted array than an unsorted array
927358  16763   How to undo the most recent commits in Git
2003505 12774   How do I delete a Git branch both locally and remotely
292357  9484    What is the difference between  git pull  and  git fetch
477816  8830    What is the correct JSON content type
231767  7737    What does the "yield" keyword do
179123  7684    How to modify existing  unpushed commits
111102  7658    How do JavaScript closures work
503093  7640    How do I redirect to another webpage
1642028 7411    What is the " >" operator in C
```

Figure 5.4: Top 10 posts by score

```
/usr/bin/hive
11227809        20737
927358  16763
2003505 12774
292357  9484    J Pablo Fern 225 ndez
477816  8830    Oli
231767  7737    Alex S
179123  7684    Laurie Young
111102  7658    e satis
503093  7640    venkatachalam
1642028 7411
```

Figure 5.5: Top 10 users by post score

```
/usr/bin/hive
165
```

Figure 5.6: Number of distinct users, who used the word "hadoop"

5.3. Calculate TF-IDF

5.3.1. Mrjob Configuration

In order to run the *MRjob* Python code, some configurations were needed. *Mrjob.conf* file was created with information related to AWS id, AWS secret key, region, name of EC2 key pair, master and cores instances types, number of instances, and subnet id as shown in Figure 5.7.

```
mrjob.conf — ~/Downloads/final
mrjob.conf
runners:
  emr:
    aws_access_key_id: AKIAIBH6GCAADBAA2KQA
    aws_secret_access_key: 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
    ec2_key_pair: EMR
    # ~/ and $ENV_VARS allowed here
    ec2_key_pair_file: /Users/Noha/.ssh/EMR.pem
    ssh_tunnel: true
    master_instance_type: 'm4.xlarge'
    instance_type: 'm4.2xlarge'
    num_core_instances: 3
    region: us-east-2
    subnet: subnet-4c0a9724
```

Figure 5.7: Top 10 users by post score

5.3.2. Run Mrjob Code

Before running the *mrjob* code, pig program was created with script to do a concatenation for the body and title in order to prepare the input for the TF-IDF part, as shown in Figure 5.8.

```
Input(s):
Successfully read 53222 records from: "s3://aws-noha/tables/part-v004-o001-r-00000"
Successfully read 54096 records from: "s3://aws-noha/tables/part-v000-o001-r-00000"
Successfully read 53582 records from: "s3://aws-noha/tables/part-v002-o001-r-00000"
Successfully read 11 records from: "s3://aws-noha/tables/part-v001-o001-r-00000"
Successfully read 54995 records from: "s3://aws-noha/tables/part-v003-o001-r-00000"

Output(s):
Successfully stored 215906 records in: "s3://aws-noha/results/res3"
```

Figure 5.8: Pig script log for concatenation step

Then, *Jupyter* notebook was used to run the code by specifying the input and output location in S3 bucket as shown in Figure 5.9

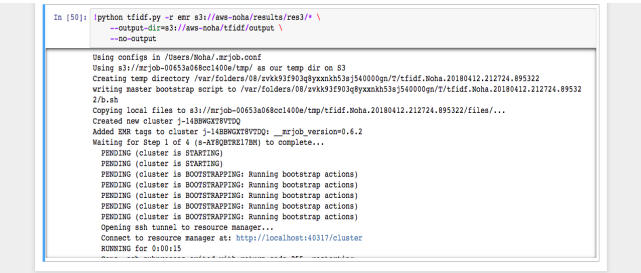


Figure 5.9: Run Mrjob code in EMR by Jupyter notebook

As a result of the previous step, a cluster was created and the four steps of the *mrjob* code were completed in 15 minutes as shown in Figure 5.10. The output of TF-IDF process produced 65 files as sown in Figure 5.11.

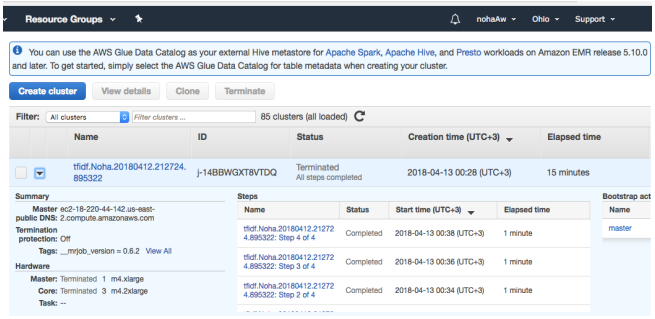


Figure 5.10: Mrjob cluster and steps

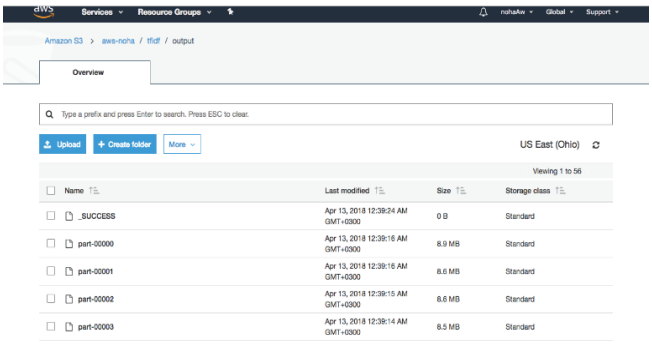


Figure 5.11: TF-IDF Output

Hive Query was implement to show the top 10 words with the highest TF-IDF scores for each user. Next figure shows Hive steps with table creation, data loading, and data querying.

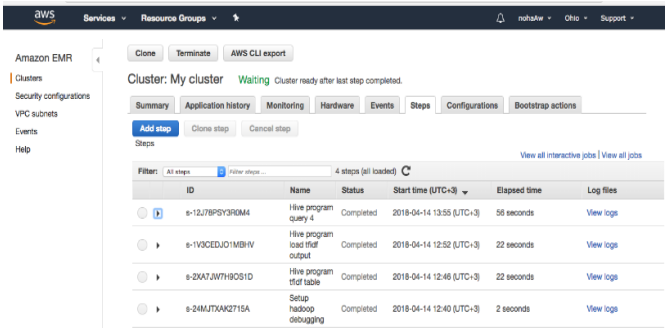


Figure 5.12: Hive for TF_IDF

Result of Hive query for top 10 words per user with highest TF-IDF scores is shown in Figure 5.13.

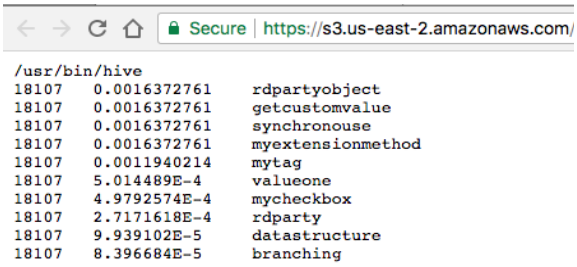


Figure 5.13: TF-IDF Hive query result

6. Problems Encountered by the Team

One of the biggest challenge is trying to know how to deal with Amazon Web Services as there are many configuration steps.

In addition, many issues were faced during running Mrjob code on Hadoop cluster. Team members tried to fix these issues in both Ubuntu VM and Cloudera VM using many suggested solutions, but the problem remains unsolved. Tried solutions were first to create temp folder manually as shown in Figure 6.1. Next, install MrJob using Pip command, as shown in Figures 6.2 and 6.3.

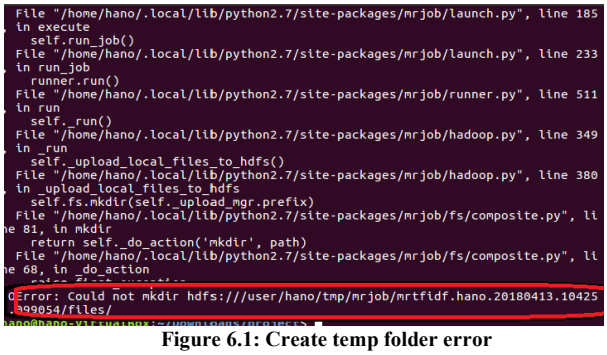


Figure 6.1: Create temp folder error

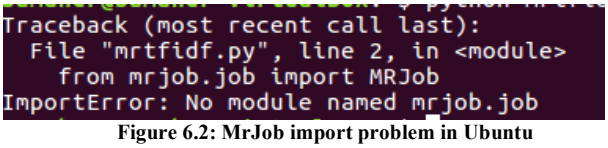


Figure 6.2: MrJob import problem in Ubuntu

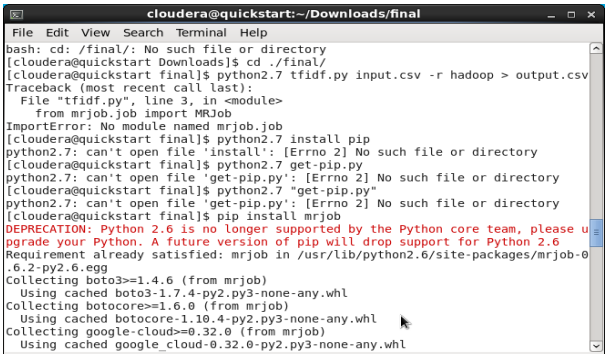


Figure 6.2: MrJob import problem in Cloudera