

# 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).

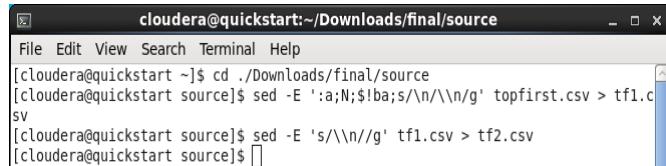
### 1. Task 1: Acquiring data

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.

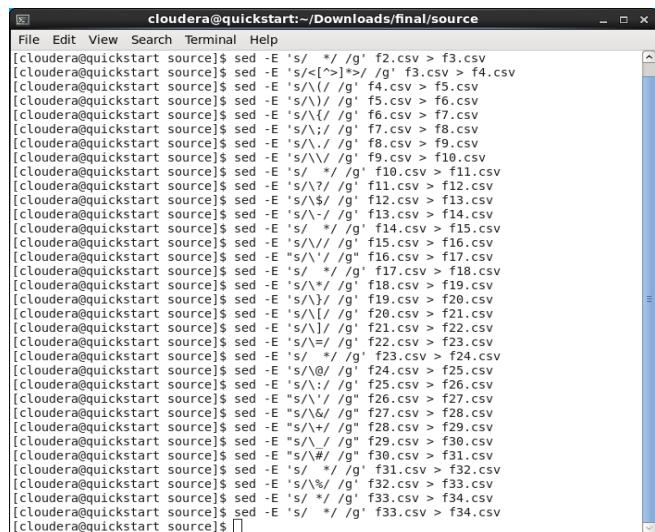
In the preprocessing step, many *sed* commands were used to keep the file clean. The first command was used to replace each new line with “/n”, while the second one was used to remove the “/n” symbol which is resulted in removing the new lines as shown on Figure 2.1.



```
[cloudera@quickstart:~/Downloads/final/source]$ cd ./Downloads/final/source
[cloudera@quickstart source]$ sed -E ':a;N;$ba;s/\n//g' topfirst.csv > tf1.csv
[cloudera@quickstart source]$ sed -E 's/\\n/g' tf1.csv > tf2.csv
[cloudera@quickstart source]$
```

Figure 2.1: Sed command on the terminal

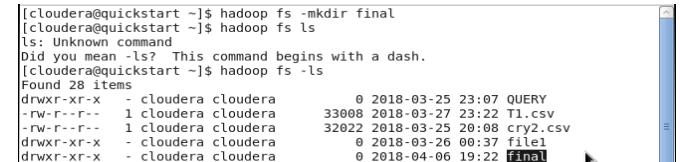
The *sed* command was used to remove punctuation and any special character except double quotation and comma to prevent changing the structure of the file. Figure 2.2 shows the list of *sed* command that has been applied on the data.



```
[cloudera@quickstart source]$ sed -E 's/ */g' f2.csv > f3.csv
[cloudera@quickstart source]$ sed -E 's/<[^>]*>/g' f3.csv > f4.csv
[cloudera@quickstart source]$ sed -E 's/(/ /g' f4.csv > f5.csv
[cloudera@quickstart source]$ sed -E 's/)/ /g' f5.csv > f6.csv
[cloudera@quickstart source]$ sed -E 's/{ /g' f6.csv > f7.csv
[cloudera@quickstart source]$ sed -E 's/;/ /g' f7.csv > f8.csv
[cloudera@quickstart source]$ sed -E 's// /g' f8.csv > f9.csv
[cloudera@quickstart source]$ sed -E 's/\\v/g' f9.csv > f10.csv
[cloudera@quickstart source]$ sed -E 's/ */g' f10.csv > f11.csv
[cloudera@quickstart source]$ sed -E 's/?/ /g' f11.csv > f12.csv
[cloudera@quickstart source]$ sed -E 's/$/ /g' f12.csv > f13.csv
[cloudera@quickstart source]$ sed -E 's/-/ /g' f13.csv > f14.csv
[cloudera@quickstart source]$ sed -E 's/ */g' f14.csv > f15.csv
[cloudera@quickstart source]$ sed -E 's// /g' f15.csv > f16.csv
[cloudera@quickstart source]$ sed -E 's/\\'/ /g' f16.csv > f17.csv
[cloudera@quickstart source]$ sed -E 's/ */ /g' f17.csv > f18.csv
[cloudera@quickstart source]$ sed -E 's/* /g' f18.csv > f19.csv
[cloudera@quickstart source]$ sed -E 's/} /g' f19.csv > f20.csv
[cloudera@quickstart source]$ sed -E 's/[ /g' f20.csv > f21.csv
[cloudera@quickstart source]$ sed -E 's/]/ /g' f21.csv > f22.csv
[cloudera@quickstart source]$ sed -E 's/= /g' f22.csv > f23.csv
[cloudera@quickstart source]$ sed -E 's/ */ /g' f23.csv > f24.csv
[cloudera@quickstart source]$ sed -E 's/@ /g' f24.csv > f25.csv
[cloudera@quickstart source]$ sed -E 's/: /g' f25.csv > f26.csv
[cloudera@quickstart source]$ sed -E 's/\\'/ /g' f26.csv > f27.csv
[cloudera@quickstart source]$ sed -E 's/&/ /g' f27.csv > f28.csv
[cloudera@quickstart source]$ sed -E 's/+ /g' f28.csv > f29.csv
[cloudera@quickstart source]$ sed -E 's/- /g' f29.csv > f30.csv
[cloudera@quickstart source]$ sed -E 's/# /g' f30.csv > f31.csv
[cloudera@quickstart source]$ sed -E 's/ */ /g' f31.csv > f32.csv
[cloudera@quickstart source]$ sed -E 's/% /g' f32.csv > f33.csv
[cloudera@quickstart source]$ sed -E 's/ */ /g' f33.csv > f34.csv
[cloudera@quickstart source]$ sed -E 's/ */ /g' f33.csv > f34.csv
[cloudera@quickstart source]$
```

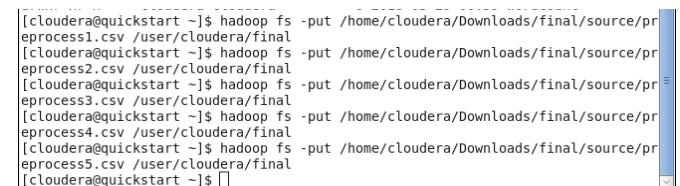
Figure 2.2: The list of sed command

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.



```
[cloudera@quickstart ~]$ hadoop fs -mkdir final
[cloudera@quickstart ~]$ hadoop fs ls
ls: Unknown command
Did you mean -ls? This command begins with a dash.
[cloudera@quickstart ~]$ hadoop fs -ls
Found 28 items
drwxr-xr-x - cloudera cloudera 0 2018-03-25 23:07 QUERY
-rw-r--r-- 1 cloudera cloudera 33008 2018-03-27 23:22 T1.csv
-rw-r--r-- 1 cloudera cloudera 32022 2018-03-25 20:08 cry2.csv
drwxr-xr-x - cloudera cloudera 0 2018-03-26 00:37 file1
drwxr-xr-x - cloudera cloudera 0 2018-04-06 19:22 final
```

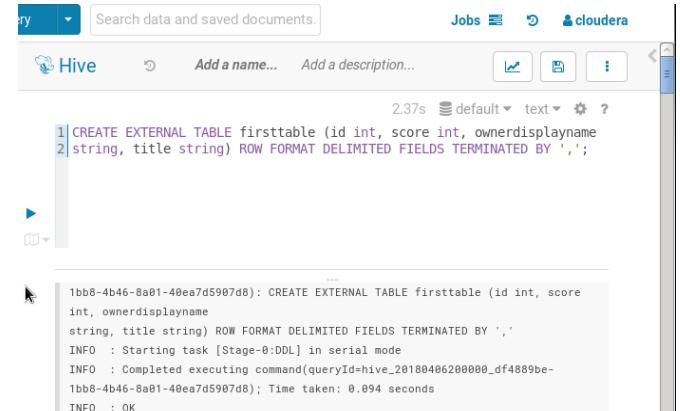
Figure 2.3-a: The identification of the folder on Hadoop cluster



```
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Downloads/final/source/procress1.csv /user/cloudera/final
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Downloads/final/source/procress2.csv /user/cloudera/final
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Downloads/final/source/procress3.csv /user/cloudera/final
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Downloads/final/source/procress4.csv /user/cloudera/final
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Downloads/final/source/procress5.csv /user/cloudera/final
[cloudera@quickstart ~]$
```

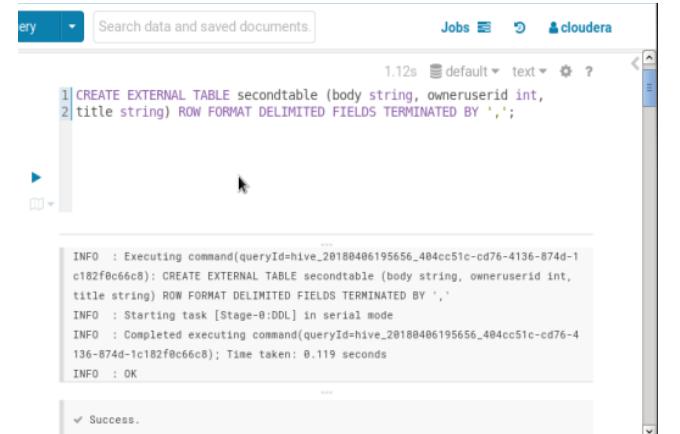
Figure 2.3-b: The identification of the files on Hadoop cluster

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.



```
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ CREATE EXTERNAL TABLE firstable (id int, score int, ownerdisplayname string, title string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Figure 2.4-a: First table creation using hive command



```
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ CREATE EXTERNAL TABLE secondtable (body string, owneruserid int, title string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

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/piggibank.jar;
2018-04-06 20:14:19,153 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-04-06 20:14:19,153 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addres
s
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*.

```
cloudera@quickstart:~
```

```
File Edit View Search Terminal Help
grunt> topfirst = LOAD '/user/cloudera/final/preprocess1.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage('','NO_MULTILINE','NOCCHANGE','SKIP_INPUT_HEADER') AS (Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreateOnDate: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:chararray, CommunityOwnedDate:datetime);
grunt> topsecond = LOAD '/user/cloudera/final/preprocess2.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage('','NO_MULTILINE','NOCCHANGE','SKIP_INPUT_HEADER') AS (Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreateOnDate: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:chararray, CommunityOwnedDate:datetime);
grunt> topthird = LOAD '/user/cloudera/final/preprocess3.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage('','NO_MULTILINE','NOCCHANGE','SKIP_INPUT_HEADER') AS (Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreateOnDate: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:chararray, CommunityOwnedDate:datetime);
grunt> topfourth = LOAD '/user/cloudera/final/preprocess4.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage('','NO_MULTILINE','NOCCHANGE','SKIP_INPUT_HEADER') AS (Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreateOnDate:datetime, DeletionDate:datetime, Score:int, ViewCount:int, Body:chararray,
```

**Figure 2.5-a: Loading some files using CSVExcelStorage**

```
grunt> topfifth = LOAD '/user/cloudera/final/preprocess5.csv' USING org.apache.pig.piggybank.storage.CSVExcelStorage(' ', 'NO_MULTILINE', 'NOCCHANGE', 'SKIP_INPUT_HEADER') AS (Id:int, PostType:id:int, AcceptedAnswerId:int, ParentId:int, CreatedOn: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: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','`'), OwnerUserId
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, ViewC
nt, REPLACE('Body','`'), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivit
e, REPLACE('Title','`'), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> T = FOREACH topthird GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewC
nt, REPLACE('Body','`'), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivit
e, REPLACE('Title','`'), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> FO = FOREACH topfourth GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewC
nt, REPLACE('Body','`'), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivit
e, REPLACE('Title','`'), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> F0 = FOREACH topfifth GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewC
nt, REPLACE('Body','`'), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivit
e, REPLACE('Title','`'), Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate;
grunt> FI = FOREACH topififth GENERATE Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewC
nt, REPLACE('Body','`'), OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivit
e, 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.

**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;
```

The screenshot shows the Cloudera Hue interface in Mozilla Firefox. The URL bar displays "quickstart.cloudera:8888/hue/editor?editor=194". The main navigation bar includes links for Cloudera Manager, Hue, Hadoop, HBase, Impala, Spark, Solr, Oozie, and Getting Started. Below the navigation is a search bar with placeholder text "Search data and saved documents...". A dropdown menu labeled "Query" is open, showing options like "HQL", "SQL", "Text", and "JSON". The main content area is titled "Results (10)". It lists ten rows of data from a query:

	id	score	title
1	11227809	20737	Why is it faster to process a sorted array than an unsorted array
2	927358	16763	How to undo the most recent commits in Git
3	2005035	12774	How do I delete a Git branch both locally and remotely
4	292357	9484	What is the difference between git pull and git fetch
5	477816	8830	What is the correct JSON content type
6	231767	7737	What does the "yield" keyword do
7	179123	7684	How to modify existing, unpushed commits
8	111102	7658	How do JavaScript closures work
9	503093	7640	How do I redirect to another webpage
10	1642028	7411	What is the "+" operator in C

**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

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

```
Select count (distinct (owneruserid) from seconddata 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/seconddata/part-m-00000' USING PigStorage(',');
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A2 = load '/user/hive/warehouse/seconddata/part-m-00001' USING PigStorage(',');
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A3 = load '/user/hive/warehouse/seconddata/part-m-00002' USING PigStorage(',');
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A4 = load '/user/hive/warehouse/seconddata/part-m-00003' USING PigStorage(',');
AS (Body:chararray, OwnerUserId:int, Title:chararray);
A5 = load '/user/hive/warehouse/seconddata/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> AA1 = FOREACH AA1 GENERATE $0,$1;
grunt> AA2 = FOREACH AA2 GENERATE $0,$1;
grunt> AA3 = FOREACH AA3 GENERATE $0,$1;
grunt> AA4 = FOREACH AA4 GENERATE $0,$1;
grunt> AA5 = 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.

```
cloudera@quickstart:~$ File Edit View Search Terminal Help
c5 = FOREACH AAAS GENERATE $0, REPLACE($1, '[0-9]*', '');
grunt> c6= UNION c1,c2,c3,c4,c5;
grunt>
grunt> store c6 into '/user/cloudera/final/tfidf_input' using PigStorage(',');
2018-04-14 17:09:02,785 [main] INFO org.apache.pig.tools.pigstats.ScriptState -
| Pig features used in the script: UNION
2018-04-14 17:09:02,959 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter], RULES DISABLED=[FilterLogicalExpressionSimplifier, PartitionFilterOptimizer]}
```

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.

```
n@n-VirtualBox:~/Downloads$ cat tfidf_input/part-m-0000* -> input.csv
```

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.

output.csv			
trackable,8062313,0.000315435219367			
trackable,8391990,0.000618039175573			
trackable,884995,0.00012496336975			
trackable,3646872,0.000618039175573			
trackable,1508493,0.000618039175573			
trackable,2440814,0.000615951205305			
trackable,2486915,0.000125539643089			
trackable,2089684,0.000626534559429			
trackable,214812,0.000483611556403			
trackable,1508493,0.000618039175573			
trackablecomputation,1505493,0.00205156356427			
trackabstractmaxlines,764099,0.000867782218537			
trackactivedirective,124069,0.000509568430737			
trackactivepart,1505493,0.001182578178214			
trackadapt,16742,0.000125539643089			
trackajsonobject,54746,0.002915958788358			
trackajsend,54746,0.00201959788358			
trackalpha,1917404,0.014846112009			
trackappopenwithremotenotificationpayload,1509580,0.00149674698729			
tracklist,1505493,0.000125539643089			
trackmax,78289,0.000125539643089			
trackback,658500,0.00196767389837			
trackback,2306977,0.00336472236621			
trackback,16742,0.00014460034225			
trackback,1352749,0.000435281030558			
trackball,1012284,0.000125539643089			
trackball,239193,0.00238703562507			
trackball,184367,0.00028098219129			
trackball,1012284,0.000513089978235			
trackballcontrols,3476710,0.00424174628829			

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/...
[cloudera@quickstart Downloads]$
```

Figure 4.8: moving the output file to Hadoop cluster

Then, Hive table was created as in Figure 4.9.

HUE			
Query		Search data and save	
<code>1 CREATE EXTERNAL TABLE table_tfidf (word string, owneruserid int, tfidf float) ROW FORMAT DELIMITED 2 FIELDS TERMINATED BY ',';</code>			32.82s default text ?
INFO : Executing command(queryId=hive_201804131316566_1e9b437-e9d4-40b8-8eef-21bf92d920): CREATE EXTERNAL			
Table created successfully.			

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.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2018-04-13 17:10:14,723 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.addrs
grunt> r = LOAD '/user/cloudera/...
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.

HUE			
Query		Search data and save	
<code>1 select owneruserid, tfidf, word 2 from table_tfidf 3 where owneruserid = 18107 4 sort by tfidf desc 5 limit 10;</code>			2m, 26s default text ?
INFO : Stage 0: Map: 1 Reduce: 1 Cumulative CPU: 6.52 sec HDFS Read: job_1523706132111_0001_s INFO : Total MapReduce CPU Time Spent: 29 seconds 659 msec INFO : Committed execution command:mrunit 7910941131444420748633574-d40fb-472b-93b0-1			

Figure 4.11: Hive query to return the top 10 terms

HUE			
Query		Search data and save	
<code>owneruserid tfidf word</code>			word
1 18107 0.0016372761456295848 rdpartyobject 2 18107 0.0016372761456295848 synchronous 3 18107 0.0016372761456295848 getcustomvalue 4 18107 0.0016372761456295848 myextensionmethod 5 18107 0.0011940214317291975 mytag 6 18107 0.0005014489288441836 valueone 7 18107 0.0004979257354441333 mycheckbox 8 18107 0.00027171618421562016 rdparty 9 18107 9.391021649352006e-05 datastructure 10 18107 8.3966842794325203e-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.

Amazon S3																									
Services > Resource Groups > nohaha > Global > Support >																									
Amazon S3 - aws-noha																									
Overview Properties Permissions Management																									
Type a prefix and press Enter to search. Press ESC to clear.																									
<a href="#">Upload</a> <a href="#">+ Create folder</a> <a href="#">More</a>																									
US East (Ohio) Viewing 1 to 6																									
<table border="1"><thead><tr><th>Name</th><th>Last modified</th><th>Size</th><th>Storage class</th></tr></thead><tbody><tr><td>data</td><td>---</td><td>---</td><td>---</td></tr><tr><td>logs</td><td>---</td><td>---</td><td>---</td></tr><tr><td>results</td><td>---</td><td>---</td><td>---</td></tr><tr><td>scripts</td><td>---</td><td>---</td><td>---</td></tr></tbody></table>						Name	Last modified	Size	Storage class	data	---	---	---	logs	---	---	---	results	---	---	---	scripts	---	---	---
Name	Last modified	Size	Storage class																						
data	---	---	---																						
logs	---	---	---																						
results	---	---	---																						
scripts	---	---	---																						

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.

The screenshot shows the AWS EMR Cluster screen with the following details:

- Clusters:** 1 step completed.
- Security configurations:** 10 steps (all loaded).
- Steps:**
  - DAUWXT7TGF Pig program Completed 2018-04-08 21:24 UTC+0 1 minute controller | syslog | stderr | stdout C View logs
  - IDBNGQPTSLQD9N Hive program Query 3 Completed 2018-04-08 18:41 UTC+0 1 minute controller | syslog | stderr | stdout C View logs
  - 3633WZDX1DNP Hive program Load 1 Completed 2018-04-08 18:27 UTC+0 26 seconds controller | syslog | stderr | stdout C View logs
  - 362ZHQQE4HFGQ Hive program Query 2 Completed 2018-04-08 18:15 UTC+0 54 seconds controller | syslog | stderr | stdout C View logs
  - ID6J9M00EERUC Hive program Query 1 Completed 2018-04-08 17:57 UTC+0 44 seconds controller | syslog | stderr | stdout C View logs
  - AWKASMD12KNT Hive program Load 1 Completed 2018-04-08 17:45 UTC+0 38 seconds controller | syslog | stderr | stdout C View logs
  - 7B8B0DMZT8RF Pig program Completed 2018-04-08 17:24 UTC+0 1 minute controller | syslog | stderr | stdout C View logs
  - MRVY3VXMYB1 Hive program2 Completed 2018-04-08 17:06 UTC+0 22 seconds controller | syslog | stderr | stdout C View logs
  - IF1GNYDFPME0X Hive program1 Completed 2018-04-08 17:04 UTC+0 24 seconds controller | syslog | stderr | stdout C View logs
  - YMP92ZNA8AKSNK Setup hadoop debugging Completed 2018-04-08 17:03 UTC+0 2 seconds controller | syslog | stderr | stdout C View logs
- Feedback:** English (US)
- Help:**

Figure 5.3: Completed steps

```
/usr/bin/hive
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 Fernández
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
runners:
  emr:
    aws_access_key_id: AKIAIBH6GCAADBAA2K0A
    aws_secret_access_key: LdxZVm6CB6tn2jatTULRuvqy5Vh+wypwfVefnLB
    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

```
In [50]: !python tfidf.py -r emr s3://aws-noha/results/res3* \ 
--output-dir=s3://aws-noha/tfidf/output \
--no-output
```

Using config in /Users/Noha/.mrjob.conf  
Using s3://mrjob-00553a08cc1400e/tmp as our temp dir on s3  
Creating temp directory /var/folders/08/vrxk3193qbxxxxb3s1540000gn/7/tfidf.Noha.20180412.212724.89532  
Writing master bootstrap script to /var/folders/08/vrxk3193qbxxxxb3s1540000gn/7/tfidf.Noha.20180412.212724.89532  
.../ah  
Copying local files to s3://mrjob-00553a08cc1400e/tmp/tfidf.Noha.20180412.212724.89532/files/...  
Creating new cluster j-14BBWGX8VTQD  
Added EMR tag to cluster j-14BBWGX8VTQD: \_\_mrjob\_version=0.4.2  
Waiting for Step 1 (j-14BBWGX8VTQD) to complete...  
PENDING (cluster is STARTING)  
PENDING (cluster is STARTING)  
PENDING (cluster is BOOTSTRAPPING) Running bootstrap actions  
Opening ssh tunnel to resource manager...  
Connecting to resource manager at http://localhost:40317/cluster  
RUNNING for 0:00:15

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 shown in Figure 5.11.

The screenshot shows the AWS EMR Resource Groups screen with the following details:

- Clusters:** 1 step completed.
- Steps:**
  - tfid.Noha.20180412.212724.89532: Step 1 of 4 completed (Name: master, Status: Completed, Start time: 2018-04-13 00:28 UTC+0, Elapsed time: 1 minute)
  - tfid.Noha.20180412.212724.89532: Step 2 of 4 completed (Name: core, Status: Completed, Start time: 2018-04-13 00:36 UTC+0, Elapsed time: 1 minute)
  - tfid.Noha.20180412.212724.89532: Step 3 of 4 completed (Name: task, Status: Completed, Start time: 2018-04-13 00:34 UTC+0, Elapsed time: 1 minute)
- Hardware:**
  - Master: Terminated 1 m4.xlarge
  - Core: Terminated 3 m4.2xlarge
  - Task: --

Figure 5.10: Mrjob cluster and steps

The screenshot shows the AWS S3 Overview screen with the following details:

- Bucket:** nohaAw
- Prefix:** /tfid / output
- Objects:**
  - \_SUCCESS
  - part-00000
  - part-00001
  - part-00002
  - part-00003
- Storage Class:** Standard
- Last modified:** Apr 13, 2018 12:39:24 AM GMT+0300
- Size:** 0 B

Figure 5.11: TF-IDF Output

Hive Query was implemented 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.

```
/usr/bin/hive
18107 0.0016372761 rdpartyobject
18107 0.0016372761 getcustomvalue
18107 0.0016372761 synchronouse
18107 0.0016372761 myextensionmethod
18107 0.0011940214 mytag
18107 5.014489E-4 valueone
18107 4.9792574E-4 mycheckbox
18107 2.7171618E-4 rdparty
18107 9.939102E-5 datastructure
18107 8.396684E-5 branching
```

Figure 5.13: TF-IDF Hive query result

## 6. Problems Encountered by the Team

One of the biggest challenges 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.

```
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/launch.py", line 185
    in execute
        self.run_job()
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/launch.py", line 233
    in run_job
        runner.run()
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/runner.py", line 511
    in run
        self._run()
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/hadoop.py", line 349
    in _run
        self._upload_local_files_to_hdfs()
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/hadoop.py", line 380
    in _upload_local_files_to_hdfs
        self._fs.mkdirs(self._upload_mgr.prefix)
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/fs/composite.py", line 81, in mkdir
    return self._do_action('mkdir', path)
File "/home/hano/.local/lib/python2.7/site-packages/mrjob/fs/composite.py", line 68, in _do_action
    ...

Error: Could not mkdir hdfs://user/tmp/mrjob/mrtfidf.hano.20180413.10425_399854/files/
hano@hano-VirtualBox:~$ cd Downloads/
```

Figure 6.1: Create temp folder error

```
Traceback (most recent call last):
  File "mrtfidf.py", line 2, in <module>
    from mrjob.job import MRJob
ImportError: No module named mrjob.job
```

Figure 6.2: MrJob import problem in Ubuntu

```
cloudera@quickstart:~/Downloads/final
bash: cd: /Final/: No such file or directory
[cloudera@quickstart Downloads]$ cd /Final/
[cloudera@quickstart final]$ python2.7 tfidf.py input.csv -r hadoop > output.csv
Traceback (most recent call last):
  File "tfidf.py", line 3, in <module>
    from mrjob.job import MRJob
ImportError: No module named mrjob.job
[cloudera@quickstart final]$ python2.7 install pip
python2.7: can't open file 'install': [Errno 2] No such file or directory
[cloudera@quickstart final]$ python2.7 get-pip.py
python2.7: can't open file 'get-pip.py': [Errno 2] No such file or directory
[cloudera@quickstart final]$ python2.7 get-pip.py
python2.7: can't open file 'get-pip.py': [Errno 2] No such file or directory
[cloudera@quickstart final]$ pip install mrjob
DEPRECATION: Python 2.6 is no longer supported by the Python core team, please upgrade your Python. A future version of pip will drop support for Python 2.6
Requirement already satisfied: mrjob in /usr/lib/python2.6/site-packages/mrjob-0.6.2-py2.6.egg
Collecting boto3>=1.4.6 (from mrjob)
  Using cached boto3-1.7.4-py2.py3-none-any.whl
Collecting botocore=1.6.0 (from mrjob)
  Using cached botocore-1.10.4-py2.py3-none-any.whl
Collecting google-cloud=0.32.0 (from mrjob)
  Using cached google-cloud-0.32.0-py2.py3-none-any.whl
```

Figure 6.2: MrJob import problem in Cloudera