Stack Exchange Data Analysis using Pig and Hive

Submitted as Assignment 1 for CA675

Name: Divya Aren
Course: *MCM 2021*
email:divya.aren2@mail.dcu.ie

***Assignment Problem*—1. To get data from Stack Exchange 2. Load them with PIG 3. Query them with Hive. 4. Calculate TF-IDF with MapReduce/Pig/Hive.**

I. Introduction

Stack Data Exchange is a question answering forum used for resolution of technical queries. Data explorer on stack exchange was used for data acquisition of two million records. Cloud Dataproc was used to create a cluster of 3 nodes (1 master and 2 worker nodes) with minimum configuration. As the number of records that can be downloaded at once is restricted to 50,000, data was downloaded data in four CSV files by writing queries. The four CSV files are loaded in Pig using CSVExcelStorage() from the Hadoop directory. CSVExcelStorage() has a feature to load multiline content from body and skip the input header. Once the data is loaded, it is transformed and then stored using PigStorage. Success logs are deleted from the Hadoop directory and the partitioned merged and cleaned file is available to be loaded into Hive for Querying. In hive, after creation of table and loading data from pig we perform the queries and results are shared for reference in below sections. Next, we calculate per-user TF-IDF using Hivemall to return the top 10 terms used by top 10 users.

II. Statement of Procedure

1. *Acquire Top 10,000 posts by Viewcount)*

We query stack exchange via the data explorer by ViewCount and extract 4 CSV files from Stack Exchange.

> *SELECT * from Posts where ViewCount> 35000 and ViewCount< 47039 ORDER BY ViewCount DESC*
>
> *SELECT count (*) from Posts where ViewCount> 47039 and ViewCount< 65887*

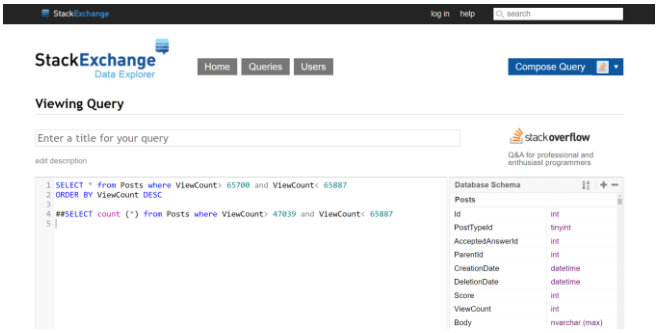

Fig 1.1: Stack Exchange Query

2. *Using Pig or MapReduce, extract, transform and load the data as applicable*

Google Cloud Platform enables Dataproc which made the process of analyzing the data fast and easy. It is hassle free and requires no complex installations of Hadoop, Hive and Pig, and used with the help of 300$ worth free credits. Only required jars were uploaded at the master node SSH and HDFS facilitated easy storing and loading of data.

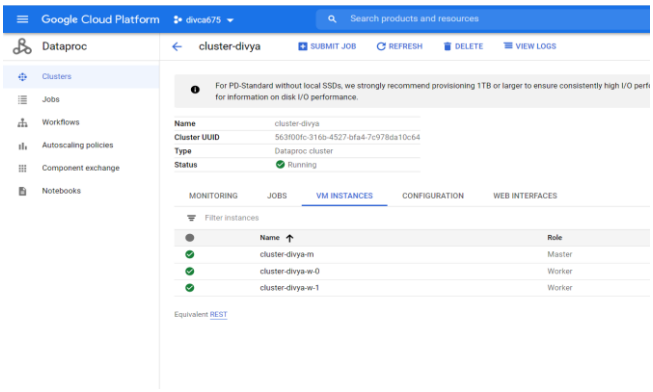

Fig 2.1: Created cluster-divya for using Hadoop, Pig and Hive

SSH terminal of master node was clicked and data was uploaded to the terminal. Data in 4 CSV files was then placed in HDFS directory using below command.

> *$ hdfs dfs -put <CSV input filename> </HDFS directory path>*

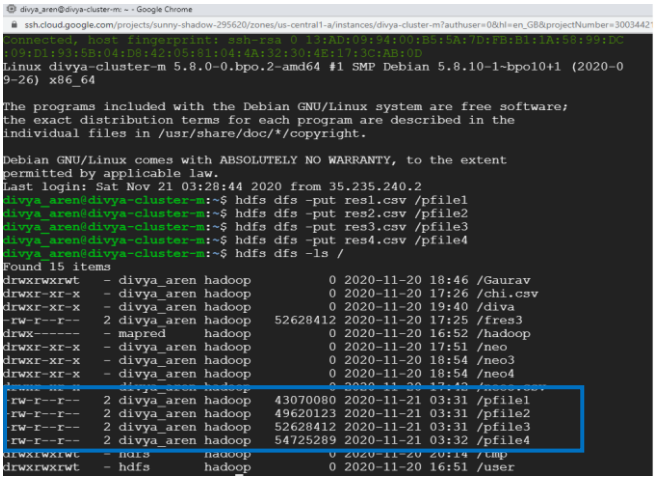

Fig 2.2: Input file placed in HDFS Directory

Pig is used to perform ETL as it is an abstraction over MapReduce and easier to perform simple transformations. PiggyBank jar which consists of many UDFs for pig was added to the directory.

Pig commands to load the data using CSVExcelLoader() with arguments like 'YES-MULTILINE' and 'SKIP_INPUT_HEADER' was used to allow multiple lines in columns like Body to be read and to skip the input header of CSV files respectively while loading the content. The four CSV files with 50,000 records each were loaded and data was merged using UNION operator, required columns were selected using FOREACH command and then cleaned for spaces, tabs, return carriage and HTML tags. FILTER command was used to eliminate 1459 records with null

values in OwnerUserId and Score column. Transformed data was then stored in HDFS directory using STORE command.

Fig 2.3.1: Commands executed in Pig

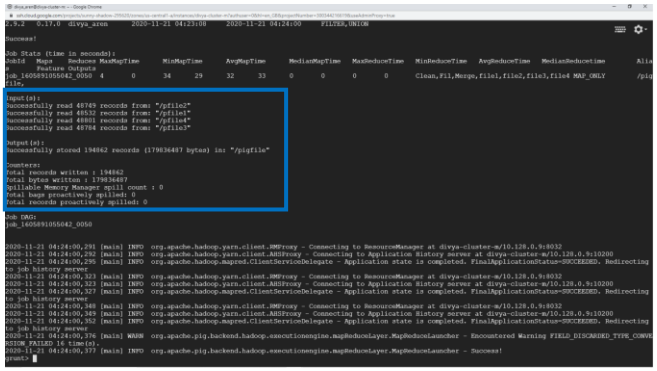

Fig 2.3.2: Results after processing and storing data in HDFS

Fig 2.4: Snapshot of transformed data in Pig

In the Hadoop directory, 4 partitioned files are created and below command was used to see the stored results from Pig.

$ hdfs dfs -ls </HDFS directory path>

Next, the success logfile is deleted from the directory to prepare to load the results in Hive.

$ hdfs dfs -rm -r </HDFS directory path/_SUCCESS>

Fig 2.6: Deleting success logs from HDFS Directory to prepare data for Hive

We may also access web-interface of the Hadoop file system created for our project in Google Cloud Platform.

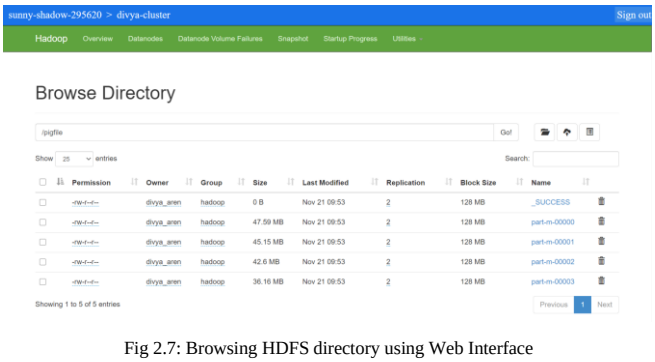

Fig 2.7: Browsing HDFS directory using Web Interface

3. *Using Hive and/or Mapreduce Query the Data applicable*

Hive is a data warehouse infrastructure tool to process and analyze data easily. The Hivecatalog jar was added for Pig and Mapreduce to access the same data structure as Hive as there is no need to convert data then. A table ‘posts’ with schema same as result from pig was created and data was loaded from HDFS directory to Hive.

Fig 3.1.: Table creation and loading data from HDFS in Hive

Fig 3.2: Snapshot of data loaded in table posts in Hive after ETL in Pig

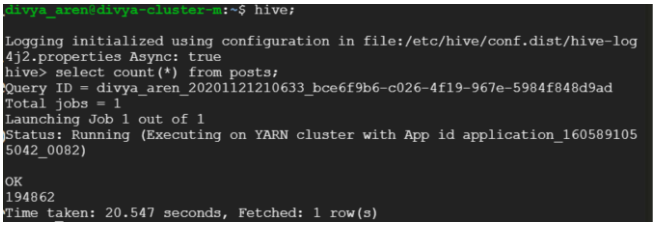

Fig 3.3: Count of records stored from Pig

3.1 The top 10 posts by score

Top 10 posts by score were queried from the above created table posts. Query and its ouput is below.

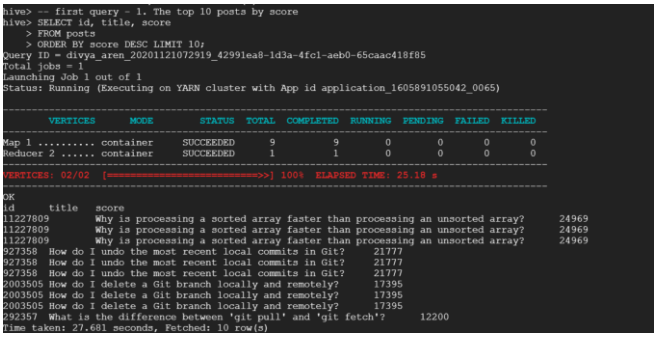

Fig 3.4: The top 10 posts by score

3.2 The top 10 users by post score

Top 10 users by post score were queried from the above created table posts. Query and its ouput is below.

```
hive> -- second query - 2. The top 10 users by post score
hive> SELECT OwnerUserId, SUM(Score) AS TotalScore
    > FROM posts
    > GROUP BY OwnerUserId
    > ORDER BY TotalScore DESC LIMIT 10;
Query ID = divya_aren_20201121073105_e2702652-a026-4bfe-8349-eb77dc5cbe81
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1605891055042_0065)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..... container     SUCCEEDED      9          9        0        0       0       0
Reducer 2 ..... container     SUCCEEDED      3          3        0        0       0       0
Reducer 3 ..... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 29.99 s
--------------------------------------------------------------------------------
OK
owneruserid     totalscore
87234   108636
4883    80799
9951    76173
6068    73713
89904   67275
51816   63807
49153   56487
95592   54873
63051   54429
179736  52083
Time taken: 31.07 seconds, Fetched: 10 row(s)
```

Fig 3.5: The top 10 users by post score

3.3 The number of distinct users, who used the word "Hadoop" in one of their posts 10 users by post score

The number of distinct users, who used the word "Hadoop" in one of their posts 10 users by post score. Query and its ouput is below.

```
hive> -- third query - 3. The number of distinct users, who used the word 'hadoop' in one of their posts
hive> SELECT COUNT(DISTINCT OwnerUserId)
    > FROM posts
    > WHERE (body LIKE '%hadoop%' OR title LIKE '%hadoop%' OR tags LIKE '%hadoop%');
Query ID = divya_aren_20201121073243_fe32a1fa-9c8c-4718-8946-68824110b55f
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1605891055042_0065)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..... container     SUCCEEDED      9          9        0        0       0       0
Reducer 2 ..... container     SUCCEEDED      3          3        0        0       0       0
Reducer 3 ..... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 28.43 s
--------------------------------------------------------------------------------
OK
_c0
525
Time taken: 29.326 seconds, Fetched: 1 row(s)
```

Fig 3.6: The number of distinct users, who used the word "Hadoop" in one of their posts 10 users by post score

Query progress, status and result can be seen on the web interface of TEZ application.

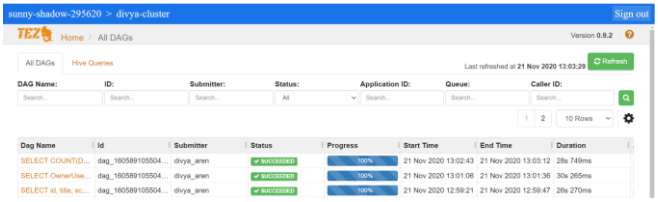

Fig 3.7: Successful execution of Hive queries on TEZ application

4. Using Mapreduce/Pig/Hive calculate the per user TF-IDF (the top 10 terms for each of the top 10 users from Query 3.2)

Per-User Term Frequency (TF) is the number of times each term occurs in all the posts by a particular user. The weight of a term that occurs in a document is simply proportional to the term frequency.

But some terms like "the", "is", 'was" etc. are common, hence calculating term frequency will tend to incorrectly emphasize documents which happen to use these words more frequently, without giving enough weight to the more meaningful terms in the posts. These common words are called stopwords and are generically defined in a UDF in Apache Hivemall.

Hence, an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely and Inverse Document Frequency (IDF) is computed using below formula.

> *TFIDF = n/N * log(D/m) n is the number of times a word is in a document N is the sum of all n's of a document*

Apache Hivemall is a collection of machine learning algorithms and versatile data analytics functions. It provides a number of ease of use machine learning functionalities through the Apache Hive UDF/UDAF/UDTF interface. Jars for Apache Hivemall and all UDFs were added in hive.

Fig 4.1: Apache Hivemall and define-All jars added to Hive

Query for top ten users according to scores was executed for reference. Columns containing text from the posts like body, tags and title are concatenated in one string for these top ten user ids.

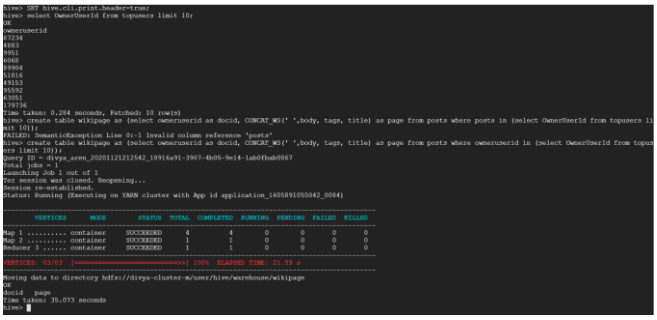

Fig 4.2: Query for Top 10 Users by Score and another query for concatenating body, tags, and title in a single string for evaluation of term frequency

A macro was created for selecting top 10 words with highest frequencies used in posts by each user along with another macro named tfidf for computation of TF-IDF using the above formula. A view wikipage_exploded was created to store the words from tokenized sentences from the corpus of body, title and tags leaving the generic stop words as defined in the hivemall jar. Another view named as term_frequency is created to store user wise term frequencies; it also calls the macro to compute the TF-IDF for all the top 10 words per user.

Fig 4.3: Creating macro and views to store exploded string and term frequency

Fig 4.4: Creating view to store term frequency of words per user

Fig 4.5: View tfidf

Query to partition the user id and words and provide TF-IDF along with TF-IDF for top 10 words by top ten users along with output snapshot are given below -

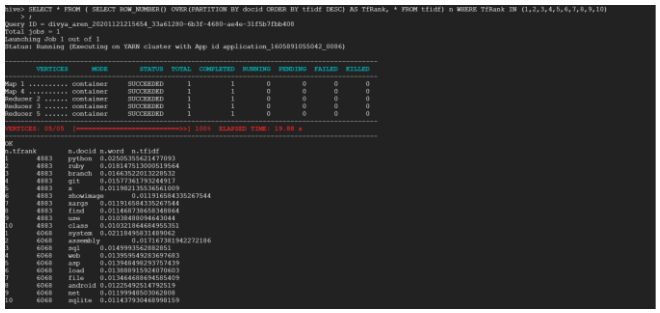

Fig 4.6: Query to partition the user id and words and provide TF-IDF for top 10 words by top ten users along with output snapshot -1

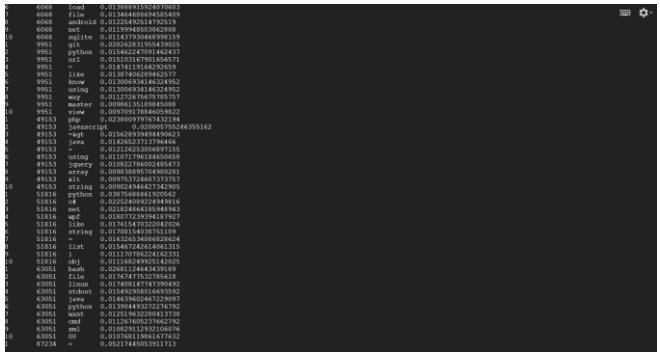

Fig 4.7: Snapshot of output -2

Fig 4.8: Snapshot of output -3

III Scope of Improvement

While performing the TF-IDF calculation there are some characters used in codes like – or = or [] etc which are part of cleaned text from the Body, Title and Tags columns. While using pig in the earlier steps we could add a FILTER to clean these data elements as well to achieve optimized top 10 terms per user.

Another way of doing it would be to add to the user-defined function stopwords() to include these characters which seems more appropriate as we are able to retain the readability of text without cleaning these special characters.

While learning about support vector models, got to learn about Okapi method to calculate the relevance of words used in documents and a comparison can be drawn between these two approaches after attempting to implement the same for a better analysis.

IV References

[1] https://pig.apache.org/docs/r0.17.0/start.html

[2] https://hive.apache.org/

[3] https://hivemall.incubator.apache.org/userguide/ft_engineering/tfidf.html

[4] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[5] Coding Blocks, TF-IDF video tutorial (downloaded video content)