Cloud Technology

Assignment 1

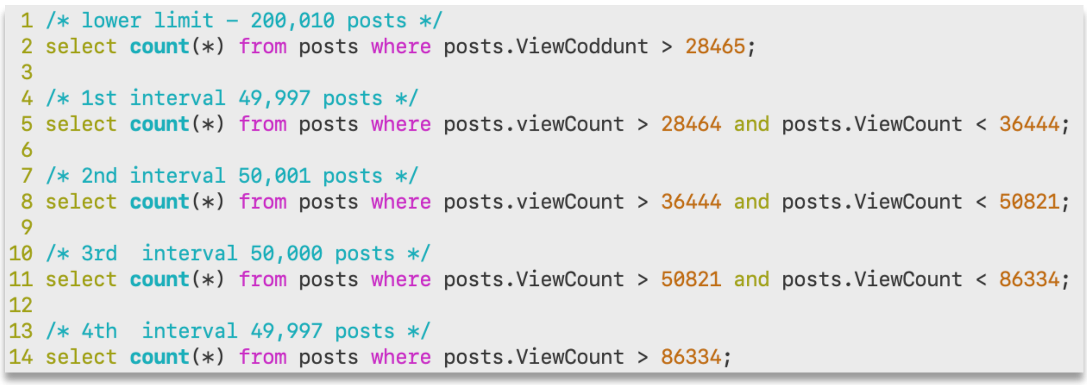
Student Name: Junli Liang
Email: junli.liang4@mail.dcu.ie
Student Number: 18212690
Github link: https://github.com/diaopk/useful_files

Introduction

This report illustrates steps on how to attempt to do the tasks. The structure of this report is divided by 4 sections corresponding to 4 individual tasks. Commands and description will be associated with each task in terms of ways to doing.

Task 1 – Acquire Post Data

Required to query 200,000 posts by ViewCount. 4 queries used to determine 4 * 50,000 posts intervals. The queried data in zip format is available https://github.com/diaopk/useful_files/blob/master/posts_v2.zip

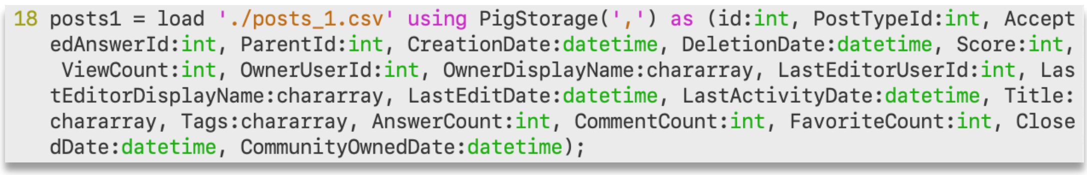


```
/* lower limit - 200,010 posts */
select count(*) from posts where posts.ViewCoddunt > 28465;

/* 1st interval 49,997 posts */
select count(*) from posts where posts.viewCount > 28464 and posts.ViewCount < 36444;

/* 2nd interval 50,001 posts */
select count(*) from posts where posts.viewCount > 36444 and posts.ViewCount < 50821;

/* 3rd  interval 50,000 posts */
select count(*) from posts where posts.ViewCount > 50821 and posts.ViewCount < 86334;

/* 4th  interval 49,997 posts */
select count(*) from posts where posts.ViewCount > 86334;
```

The numbers used to filter the ViewCount are chosen by multiple trials.

Task 2 – Use with Pig to Load/Extract Data

With Pig, I am able to load the raw data and extract it depending on the actual needs. The entire query, pig, hive commands are available at https://github.com/diaopk/useful_files/blob/master/sql_pig_hive_script.sql. With the following Pig command, queried data can be loaded into Pig. It defines a schema for loading data and specifies "," format.



```
posts1 = load './posts_1.csv' using PigStorage(',') as (id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreationDate:datetime, DeletionDate:datetime, Score:int, ViewCount:int, 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);
```

As there are 4 different csv files I needed to load in so the command above would be run 4 times. `posts = UNION posts1, posts2, posts3, posts4;` command is used to combine 4 loaded data. `FILTER posts BY (Body MATCHES '.*hadoop.*');` command is used to filter posts with the word "Hadoop" is included in Body column. This command is useful for task 3.3 described in the assignment description. Eventually the result can be stored as files using the command below.

```
STORE filter_posts INTO '<pig_output>' USING PigStorage(',');
```

Task 3 – Use with Hive to Query and Query Result Screenshots

This section uses the files generated by Pig in the previous section in Hive. First of all, I need to create a table for posts/filter_posts. In total there are 22 columns of data.

```
CREATE TABLE IF NOT EXISTS posts
(id INT, PostTypeId INT, AcceptedAnswerId INT, ParentId INT, CreationDate TIMESTAMP, DeletionDate TIMESTAMP, Score INT, ViewCount INT, Body STRING, OwnerUserId INT, OwnerDisplayName STRING, LastEditorUserId INT, LastEditorDisplayName STRING, LastEditDate TIMESTAMP, LastActivityDate TIMESTAMP, Title STRING, Tags STRING, AnswerCount INT, CommentCount INT, FavoriteCount INT, ClosedDate TIMESTAMP, CommunityOwnedDate TIMESTAMP)
COMMENT 'Posts of Stackover Exchange' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE [LOCATION <input-path>];
```

- ### Task 3.1 Query the top 10 posts by score.

  Query:

  ```
  SELECT id,score FROM posts ORDER BY score DESC LIMIT 10;
  ```

  Result shown PostId and Score:

  ```
  OK
  11227809        22586
  927358  19063
  2003505 14715
  292357  10726
  477816  9517
  231767  8943
  1642028 8112
  348170  7894
  503093  7733
  179123  7676
  Time taken: 7.516 seconds, Fetched: 10 row(s)
  ```

- ### Task 3.2 Query the top 10 users by post score.

  Query:

  ```
  SELECT OwnerUserId AS user_id, SUM(score) AS scores FROM posts
  WHERE OwnerUserId IS NOT NULL GROUP BY OwnerUserId ORDER BY scores DESC
  LIMIT 10;
  ```

  Result shown OwnerUserId and combined Score:

  ```
  OK
  87234   32477
  4883    22784
  9951    22628
  6068    21507
  89904   19731
  51816   16542
  49153   15644
  95592   15409
  63051   14858
  39677   14794
  Time taken: 6.405 seconds, Fetched: 10 row(s)
  ```

- Task 3.3 Query the number of distinct users, who used the word “Hadoop” in one of their posts. As the data already is filtered by Pig, then that is relatively easy task with Hive.

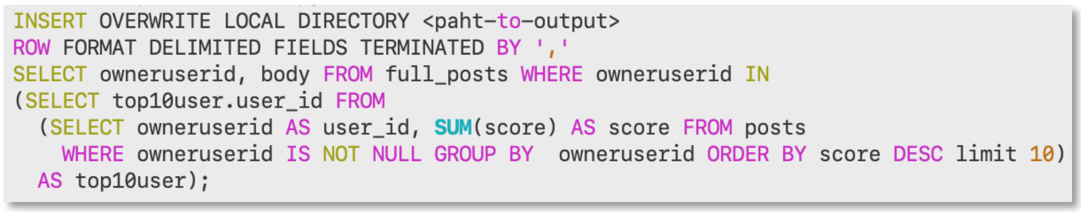
  Query:

  ```
  SELECT COUNT(DISTINCT OwnerUserId) FROM filter_posts WHERE OwnerUserId IS NOT NULL;
  ```

  Result:

  ```
  OK
  88
  Time taken: 1.809 seconds, Fetched: 1 row(s)
  ```

- Task 3.4 Pre-task of Task 4 – generate a csv format file for MapReduce program. With the command below, I can filter out top 10 users ordered by score and associated Body content. This query concludes subquery, which basically the query result from task 3.2. Then Extract top 10 OwnerUserId order by score and their Body content to generate file.



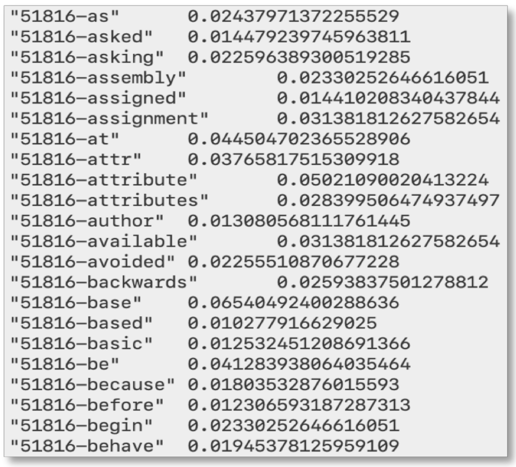
```
INSERT OVERWRITE LOCAL DIRECTORY <paht-to-output>
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
SELECT owneruserid, body FROM full_posts WHERE owneruserid IN
(SELECT top10user.user_id FROM
  (SELECT owneruserid AS user_id, SUM(score) AS score FROM posts
    WHERE owneruserid IS NOT NULL GROUP BY  owneruserid ORDER BY score DESC limit 10)
  AS top10user);
```

Task 4 – Use with MapReduce

The python MapReduce script is available at https://github.com/diaopk/useful_files/blob/master/mr_count.py. The idea behind is that I used two python datastructure Dictionary to record the TF values and IDF values respectively.

```
tf_dict is defined in { user_id: { word_a: TF_value,
                                   word_b: TF_value,
                                   … }
                      }
idf_dict is defined in
        { user_id: [ total_number_of_docs, { word_a: [num_doc_with_word_a, idf_value],
                                             word_b: [num_doc_with_word_b, idf_value],
                                             … }
                   ]
        }.
```

Mapper contains three components, mapper_init(), mapper_middle() and mapper_final(). Mapper_init() initialises two data structures. Mapper_middle() concurrently share the two defined dictionaries and update values such as total number of documents, number of documents with 'word_a/b/…' in it, etc. Mapper_final() passes user id plus a word, separated by ' – ', and a calculated TF-IDF value as tuple (e.g. (123456-how, 0.xxxx))to combiner(). There is only one step MapReduce job, which may not fully utilise the power of MaperReduce. The partial result shown below is user id (51816-<word>, tf_idf_value).



```
"51816-as"      0.024379713722555529
"51816-asked"   0.014479239745963811
"51816-asking"  0.022596389300519285
"51816-assembly"        0.02330252646616051
"51816-assigned"        0.014410208340437844
"51816-assignment"      0.031381812627582654
"51816-at"      0.044504702365528906
"51816-attr"    0.03765817515309918
"51816-attribute"       0.05021090020413224
"51816-attributes"      0.028399506474937497
"51816-author"  0.013080568111761445
"51816-available"       0.031381812627582654
"51816-avoided" 0.02255510870677228
"51816-backwards"       0.02593837501278812
"51816-base"    0.06540492400288636
"51816-based"   0.010277916629025
"51816-basic"   0.012532451208691366
"51816-be"      0.041283938064035464
"51816-because" 0.01803532876015593
"51816-before"  0.012306593187287313
"51816-begin"   0.02330252646616051
"51816-behave"  0.01945378125959109
```

Limitation

As shown, the MapReduce script shows all users and their word TF-IDF values. Results are not displayed in order and not able to show top 10 word-tfidf values, yet. Additional works need to be done in combiner and reducer.