CS 6240: Project Final Report

Team Members: Sarika Akula, Swetha Jeyaraman, Mukul Sharma

Introduction:

We worked with the Twitter datasets that has 3 different datasets tweets, network and user profiles data. We analyzed the data sets and found interesting correlations between the datasets.
The correlations that we found in the datasets are:

1. Userid and followerids
2. Userid and retweet count

Task 1: We cluster on location parameters that is latitude, longitude using kmeans clustering and then identify bots in each cluster comparing their follower count and user name length against the average of these parameters for each cluster.
Task 2: We are getting the follower count of each user and then taking location as the criteria, we get the total follower count for that particular location.We also try to find the average status count and friend count for each location in the United States.
Use of this analysis: Organizations can use this analysis to compare locations based on their follower count, get an idea of the average status count and average friend count in each location. This information would help them to identify locations with high twitter users and where it is growing.

Task 3: We worked using the three datasets to determine the character of the followers on twitter using data like retweet count, follower count, status count, friend count and favourite count. By working on this join we explored the various optimizations that Pig applies to the pig scripts while execution. We also tried switching off certain optimizers which we believed to make a lot of impact on the performance of the pig script. We compared the performance and have analysed the impacts of the optimizers.

Data:

1. **Tweets**

This contains a list of files with its name consistent with the user ID. Each file contains at most 500 tweets published by the user. The tweets are stored in the following format
We have converted this dataset into a CSV file. Please find below a list of records from this data set

100002112,status, Here's link to listen live to our discussion of #debtceiling #politics , Here's link to listen live to our discussion of , http, 96944336150867968, Fri Jul 29 09, 0, false, , debtceiling politics,

2. **UserProfile**

The user profile data set contains many useful information like the user ID, username, friend count, follower count,status count, favorite count,account age and user location in the following format

We converted this file into CSV format as well. One challenge that we had faced with this data set was that the location information was random with values like “neverland” and “under your bed”.
In order to use location for our task, we used the a Zipcode dataset which contained the city, state, lattitude and longitude values for many locations in the US. We randomly matched a record from Zipcode dataset with the user profile dataset and included the location information.

A sample dataset from the user profile dataset that we are using

100008949,esttrellitta,264,44,6853,0,28 Dec 2009 18:01:42 GMT,Megargel,AL,-87.435279,31.361214

3. **Network**

It contains a single text file with “In Twitter, a following relationship is from a user A to a user B, where A is called a follower of B, and B is called a friend of A. In the file, each line represents a following relationship from a user to another user. Specifically”

We converted this into a CSV format and the dataset looks like

10001882,10001882
100039241,100039241

Source of the data set : https://wiki.cites.illinois.edu/wiki/display/forward/Dataset-UDI-TwitterCrawl-Aug2012

Major Task 1: Major Task 1 : K-Means Clustering
Purpose of the task:
In this major task we process the users data from the twitter dataset to differentiate bots from genuine users by using their location and user-name length. To achieve this we first cluster the user using k-means then we apply filters and sorting to come up with probable bots.

Main Idea:
We Cluster all the user data on the basis of timezone by using location coordinates (Latitude and Longitude) as the clustering parameters. Initially we take the centroids as the latitude, longitude pairs in the 4 different timezones in united states, the kmeans clustering then converges to more accurate centroid coordinates.

The converged centroids calculated above are then used in the following helper tasks :
Helper Task 1 :
This task takes the centroids calculated from the k-means and then creates a new dataset containing each record with its closest centroid.

Helper Task 2 :
In this task we compare each record's user name length in each cluster with the cluster average and calculate the difference between the two. Secondly we do secondary sorting on this diffrence to determine the probable bots for each cluster.

Pseudo Code relevant:
Kmeans :
- As we are clustering on the basis of timezones so we know that k=4 hence it is not relevant to try different values of k for this problem statement.
- The initial initial 4 centroid values are selected as the location point of a place in that particular time zone.

Map -
1. Get initial list of centroid according to the four time zones from the centroid file and consider them as the initial cluster centroids
2. Fetch Latitude, longitude(location point pair), username from the data set
3. Compare each value of latitude, longitude from the dataset with each centroid values
4. Calculate the eucledian distance of each location point in data set from the cluster centroid
5. Associate the number of the nearest cluster with the elements from the dataset
6. Emit the key,value pair with the key being the cluster number and the value being the location point pair and the username of the record.

Reduce :
1. Calculate the average of the latitude and longitude in the reducer using the location point pairs for each cluster
2. Emit the key,value pair with the key being the cluster number and the value being the combination of average latitude, longitude and average username length

[Reference : http://www.geomidpoint.com/calculation.html]

Convergence Condition in Driver program :

1. We keep track of all the calculated centroids in each iteration
2. Determine the difference between the previous and new centroid values in two successive iterations
3. If the difference is < 0.1 then we say that we now have the final centroid values

Helper Tasks :

- First helper task takes the centroids calculated from kmeans and creates the dataset of all location points from the dataset with the closest cluster number
- Second helper task works on dataset created from first task and calculates the difference between each user name length and follower count with the average of these parameters calculated for each cluster, we then employ secondary sorting to arrange these records so that the users with the highest username length come up the in the list for their timezone

  Secondary Sort Pseudocode :

  ```
  Map
      1. Calculate the difference between the average value of attribute for a particular cluster of a timezone
         and the value from the dataset.
      2. Create key as the combination of cluster number and the difference calculated in the previous step.
         The value is the combination of other parameters like username, follower count, Username length
         and cluster number
      3. Emit (key, value)


  Reduce
      1. Setting the number of reduce tasks to 4 makes sure that we have a separate list for each time zone
      2. The reducer gets all the values sorted in decreasing order of the difference from the average for
         each value in dataset
      3. Emit (key, values) (Where both key and value is the same as that in the map)
  ```

Technical discussions and Comparisons:
Execution time:

Machines	**Execution time**
1 + 5 Machine cluster	**977 Seconds**
1 + 10 Machine cluster	**949 Seconds**

Execution time for Helper task 2 :

Machines	**Execution time**
1 + 5 Machine cluster	**89 Seconds**
1 + 10 Machine cluster	**85 Seconds**

Sample final output of Helper Task 2 :

Bots Identification in each cluster -
For Cluster 0 -

```
0,-9	0,955.1467997221025,-9.30056275465289,ESPNNFLSourcestellES
```

for Cluster 1 -

```
1,-9	0,1334.1467997221025,-9.30056275465289,KFCtimemiamiamjenpre
```

for Cluster 2 -

```
2,-9	0,1348.1467997221025,-9.30056275465289,WhatstdratesIhaveunl
```

for Cluster 3 -

```
3,-7	0,1336.1467997221025,-7.300562754652891,mfletcherchristian
```

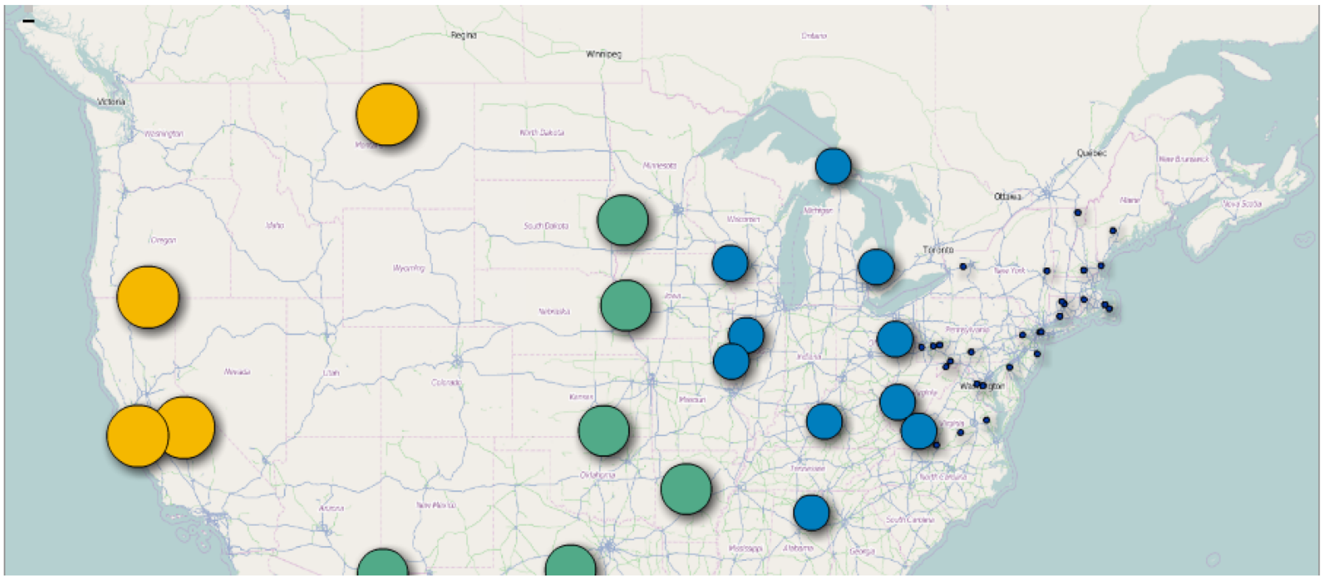
Files attached :
Kmeans-
Controller.txt and syslog.txt for 6 xLarge machine cluster
Controller.txt and syslog.txt for 11 xLarge machine cluster
kmeans.java: Source code for kmeans program

Helper task 1-
BotDetect.java : source file to create the new dataset with cluster number associated with each record

Helper task 2-
Controller.txt and syslog.txt for 6 xLarge machine cluster
Controller.txt and syslog.txt for 11 xLarge machine cluster
identifyBot.java : source file to execute secondary sorting on dataset and determine the bots
Secondarysorting files : AverageDelayKeyComparator.java, AverageDelayKeyGroupingComparator.java, AverageDelayKeyPartitioner.java, CompositeKey.java.

Machine Configurations Impact
The execution times mentioned above for the kmeans program clearly show that the program scales well when the number of machines is increased and hence the execution time is improved.

Charts:



Setup Challenges: The main challenge faced during this task was to run it on aws, We first tried running it without using distributed file cache and were able to run it successfully on local however it refused to go for more than 2 iterations on aws. So we had to use distributed file cache to run it on aws.

Conclusion:

- For KMeans we discovered that the number of iterations required and the accuracy for centroids calculated can be greatly affected by the initial centroid values that we feed to the program.

Future work :

- The current implementation to identify twitter bots just shows a part of solution needed. We can employ more machine learning techniques to trace the pattern of bot activities like the way they tweet.
- We can take help of timezones to determine user sleep patterns and compare this against each users sleep pattern.

Major Task 2: Hive Vs Plain Map Reduce
Analysis: We are to get the total follower count for each location in the Userprofile table by implementing a reduce side join of Userprofile and Network datasets.

Main Idea:
We have two datasets Userprofile and Network. The common attribute in these 2 files is the userid. We join on userid and get the total count of followers from each location (group by location).
We implement the same computation in hive and compare the results from map reduce and hive programs.

Purpose of the task: Organizations can use this analysis to compare locations based on their follower count, get an idea of the average status count and average friend count in each location.

Pseudo Code relevant:
First Map-Reduce job:
We have defined two mappers to work on Userprofile and Network data files separately. The mappers emit values with flags associated with them.

```
UserfileMapper(object key, text output) {
Parse through each row of the Userprofile Data file and emit Userid as the key and location as the value.
Arg1: Userprofile input directory
Arg2: Network input directory
Arg3: Intermediate output directory
Arg4: Final output directory
Arg3 and Arg4 directories should not be available before execution.
User profile file  contains userid, followercount, favoritecount,location, latitude and longitude fields.
userid(data[0]) and location(data[1])
Filtering the data to satisfy the condition that  userid and location are not null.
Emit (userid, location + “userprofile”)
//userprofile is  the flag, so that we can use this to differentiate records in reducer.                    }

NetworkMapper(object Key, text output) {
Network file contains information user’s followers.The data is in the format of Userid, follower id. Parse each record in network file to emit the userid as the key and follower id as the value with an additional  flag “network”.
This flag  is necessary to identify the records in reducer.
Capture userid (data[0] ) and followerid(data[1])
```

Filtering the data to satisfy the condition that userid and folllowerid are not null.
Emit (userid key, followerid + “network”)
//network is used as a flag, as similar to the above mapper

Both the above mappers emit the same key and we used a single reduce task to join the results from Userprofile and Network Mappers.
Reducer(Userid Key, location(or)followerid) {
Based on the flag from the map’s output , we segregate the information and capture the count of followers for each location.
Emit (Userid key, list of location and count); }
The above reduce phase writes output to a temporary file or an intermediate output file. We then use the intermediate file and perform another Map Reduce job to get the desired data.
Second Map Reduce Job:
LocationMapper(object Key, Text output) {
Read each line from the intermediate output file and capture location and count information.   Emit (location, count);
LocationReducer(Text key, text output) {
From the mapper, we get a list of counts for each location. We then calculate the total count for each location. We also find the average friend count an average status count.
Driver program:
The above said process runs from the driver program. We need 4 command line arguments to start the job:

Sample Output Lines:
Intermediate output:

```
100001479  Shallowater, 0, 1, 0
100010334  Durant, 0, 35, 10
100013967  Camp Hill, 45, 323, 1269
100025559  Denver, 0, 91, 15
100089301  Weston, 57, 3350, 450
```

Final sample output:

```
Abercrombie        1420,953.0,1632.0
Abingdon           5607,539.0,2136.0
Aguanga            2424,518.0,1607.0
Alcester           1146,1064.0,1342.0
Alden              12650,595.0,2236.0
Aliceville         1235,876.0,2871.0
```

Hive Program:

```
CREATE EXTERNAL TABLE userprofile_details(userid String,username String,friendcount int,followercount int,
statuscount int,favoritecount int,date int,location String,state String,longitude String,latitude String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',' stored as textfile location 's3://mrprojectsarika/hive/sampleusers';

LOAD DATA INPATH 's3://mrprojectsarika/samplenew.txt' OVERWRITE INTO TABLE userprofile_details;
```

```
//we created an external table userprofile_details with userid, username, friendcount, followercount, favoritecount,
statuscount, date, location, state,longitude,l atitude as columns and loaded the dataset into it.


CREATE EXTERNAL TABLE network_details(followerid String, userid String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',' stored as textfile location 's3://mrprojectsarika/hive/samplenetwork';
//we created an external table network_Details  with followerid and userid as columns and loaded the dataset into it.

LOAD DATA INPATH 's3://mrprojectsarika/network_0.txt' INTO TABLE network_details;

//created an external table to store intermediate join output and we join on userid.

CREATE EXTERNAL TABLE intermediate_one(location String, friendcount int,statuscount int, followerid String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',' LOCATION 's3://mrprojectsarika/hive/intermediate_one';

INSERT OVERWRITE TABLE intermediate_one
select userprofile_details.location, userprofile_details.friendcount, userprofile_details.statuscount,
network_details.followerid from userprofile_details join network_details on (userprofile_details.userid =
network_details.userid);


CREATE EXTERNAL TABLE intermediate_second_now(location String, followercount int, friendcount int,statuscount
int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION
's3://mrprojectsarika/hive/intermediate_second_now';

//Performing all the computations for followercount, average  status count and average friend count grouped by
location.

INSERT OVERWRITE TABLE intermediate_second_now
select location, count(*) as totalfollowercount, avg(friendcount) as avgfriendcount,avg(statuscount) as avgstatuscount
from intermediate_one group by location;

SELECT intermediate_second_now.location, intermediate_second_now.followercount,
intermediate_second_now.friendcount, intermediate_second_now.statuscount FROM intermediate_second_now;
```

Sample output:

```
Abercrombie 1420,953,1632
Abingdon 5607,539,2136
Aguanga  2424,518,1607
Alcester   1146,1064,1342
Alden  12650,595,2236
Aliceville  1235,876,2871
```

Files attached :

Plain map reduce
Controller.txt and syslog.txt for 5 Large machine cluster
Controller.txt and syslog.txt for 10 Large machine cluster

Controller.txt and syslog.txt for 13 Large machine

Hive
Controller.txt and syslog.txt for 5 Large machine cluster
Controller.txt and syslog.txt for 10 Large machine cluster
Controller.txt and syslog.txt for 13 Large machine

Setup Challenges:
Setting up Hive on local system
We faced a lot of challenges while installing hive on ubuntu.
There were a lot of errors that came up during installation. Eg: Hive metastore service was not starting with this command.

```
sudo service mysqld start
```

We also faced few errors because we did not install the sql driver initially.

AWS Challenges for hive:

In AWS, we took time to figure out that external tables had to be created.

We also faced an issue while running our hive script on AWS, because one of the datasets size was 6 gb which threw us an error.

Error: AWS Error Message: The specified copy source is larger than the maximum allowable size for a copy source.

The largest object that can be uploaded in a single PUT is 5 gigabytes. For larger objects,customers should consider using the Multipart Upload capability.
So, we used a 2GB input file to run the script.

Hive configuration - large machines, hadoop 2.4.7, hive 0.11
Plain map reduce - large machines,hadoop 2.4.7

Technical discussions and comparisons

Execution times:

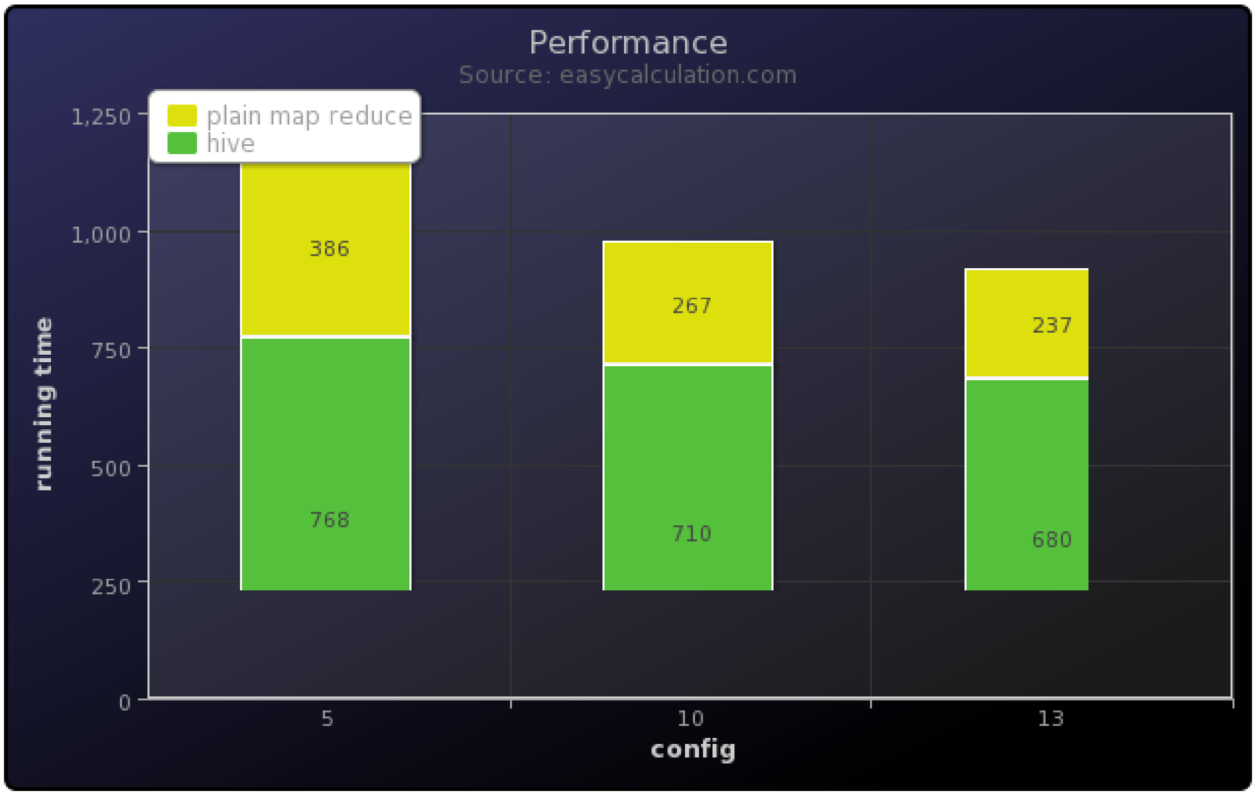
	5 machines	**10 machines**	**13 machines**
Hive	768 seconds	710 seconds	680 seconds
Plain MapReduce	386 seconds	267 seconds	231 seconds

It can be clearly seen from the results above that plain MapReduce performs faster. Hive takes time because it has to internally convert to plain MapReduce and it has to create tables.

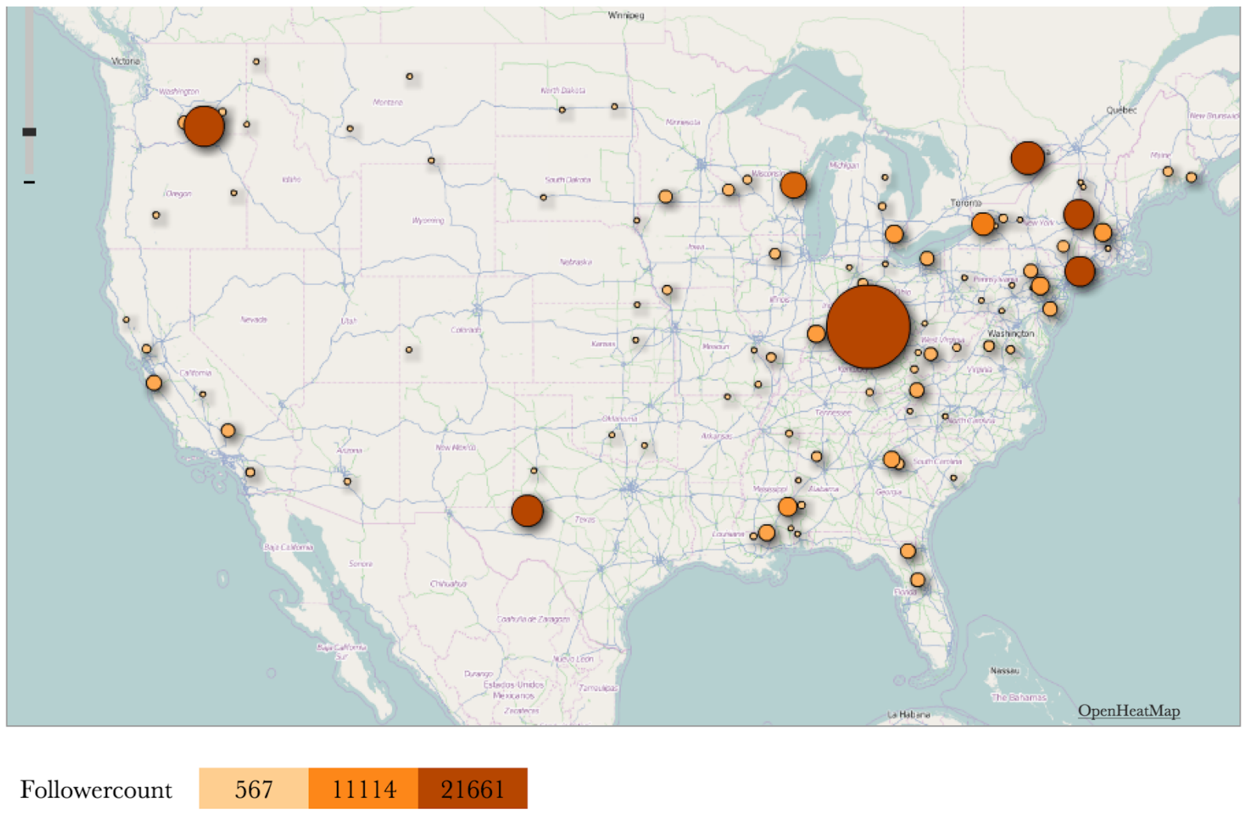
Machine Configurations Impact
The execution times mentioned above for the programs clearly show that the program scales well when the number of machines is increased and hence the execution time is improved.

Charts

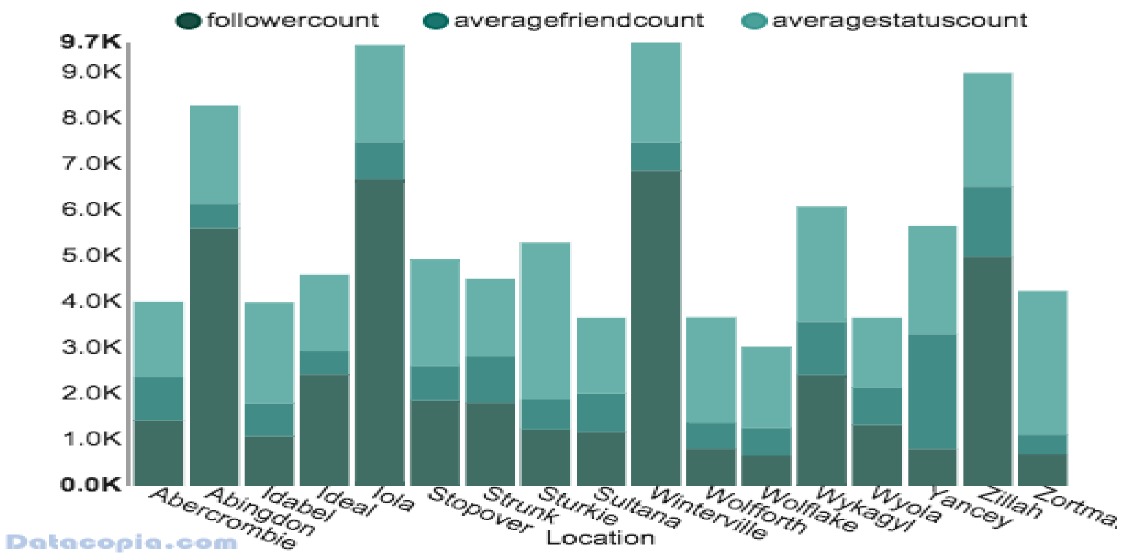
-- The chart below depicts the scalability of hive program. The running time reduces as the number of worker machines increase.

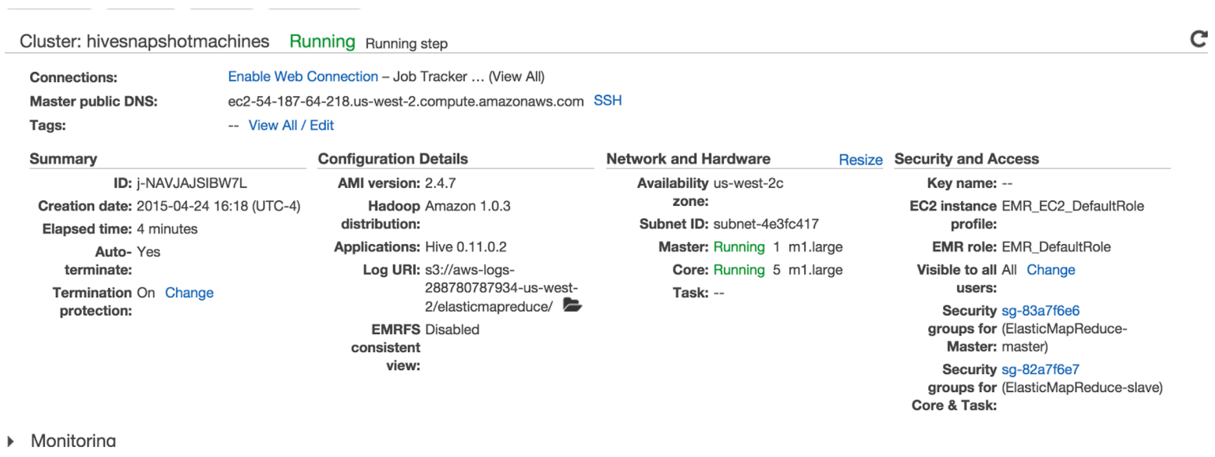


This chart below depicts the follower count based on different locations in United states.



-- The chart below depicts the sample output: Locations and their respective follower count, average status count and average friend count.




Cluster: hivesnapshotmachines Running Running step

Connections: Enable Web Connection – Job Tracker ... (View All)
Master public DNS: ec2-54-187-64-218.us-west-2.compute.amazonaws.com SSH
Tags: -- View All / Edit

Summary
ID: j-NAVJAJSIBW7L
Creation date: 2015-04-24 16:18 (UTC-4)
Elapsed time: 4 minutes
Auto-terminate: Yes
Termination protection: On Change

Configuration Details
AMI version: 2.4.7
Hadoop distribution: Amazon 1.0.3
Applications: Hive 0.11.0.2
Log URI: s3://aws-logs-288780787934-us-west-2/elasticmapreduce/
EMRFS consistent view: Disabled

Network and Hardware Resize
Availability zone: us-west-2c
Subnet ID: subnet-4e3fc417
Master: Running 1 m1.large
Core: Running 5 m1.large
Task: --

Security and Access
Key name: --
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Visible to all users: All Change
Security groups for Master: sg-83a7f6e6 (ElasticMapReduce-master)
Security groups for Core & Task: sg-82a7f6e7 (ElasticMapReduce-slave)

Monitoring

Conclusion:
We have learnt to write Hive scripts for map reduce jobs and were also successful in running these scripts on our local machine and AWS. We have not done Hive assignments earlier in the course, so it was a good learning experience to learn Hive from scratch. It can be clearly seen from the results above that plain MapReduce performs faster. Hive takes time because it has to internally convert to plain MapReduce and it has to create tables.

Major Task 3: Pig Optimization
Purpose of the task:
We worked on determining the effect of the various Pig Optimizers by generating pig scripts to solve a problem using various execution plans.
The main pig optimizations tested includes effects of Join Order, MultiQuery Optimization,Split Optimization, Push Up Filter Optimization and effects of early Projection and Selection

Main Idea:
We are trying to determine the character of the twitter followers in various location within United States . We determine the maximum retweets, average friend count, average favorite count, average follower count and average favorite count of the users who are followers of other twitter users.
This is achieved by joining UserProfile dataset with Tweets dataset. The Network dataset is then joined with the previous join's results. The join result is then split according to time zones in US and for each split the character is determined by applying the aggregation operations.

Pseudo Code:

```
REGISTER file:/home/hadoop/lib/pig/piggybank.jar
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;
SET default_parallel 10;

--Load the UserProfile dataset
users= LOAD 's3://mrprojectsarika/samplenew.txt' using PigStorage(',') AS
(userId,userName,friendCount:int,followerCount:int,statusCount:int,favoriteCount:int,accountAge,city,state,longitude,l
atitude);

-- Project Users table so that we can get a positive longitude value
```

```
projectUser= FOREACH users GENERATE
userId,userName,friendCount,followerCount,statusCount,favoriteCount,state,-longitude AS timezone;

--Load the Networks dataset
networks= LOAD 's3://mrprojectsarika/network.txt' using PigStorage(',') AS (followerId, friendId);

--Load the Tweets dataset
tweets= LOAD 's3://mrprojectsarika/tweets.txt' using PigStorage(',') AS (tuserId, status,originalText,
copyText,link,tweetId,time,retweetCount:int,favorite,mentionedUserId,hashTags);

-- Filter the User table based on longitude into different time zones
filterUsersCentral= FILTER projectUser BY timezone >=75 AND $7< 105;
filterUsersEast= FILTER projectUser BY timezone <75;
filterUsersMountain= FILTER projectUser BY timezone >=105 AND $7< 120;
filterUsersPacific= FILTER projectUser BY timezone >=120;

-- Project the filtered UserProfile, networks and tweets data set
projectedUsersCentral= FOREACH filterUsersCentral GENERATE
userId,userName,friendCount,followerCount,statusCount,favoriteCount,state,timezone;
projectedUsersEast= FOREACH filterUsersEast GENERATE
userId,userName,friendCount,followerCount,statusCount,favoriteCount,state,timezone;
projectedUsersMountain= FOREACH filterUsersMountain GENERATE
userId,userName,friendCount,followerCount,statusCount,favoriteCount,state,timezone;
projectedUsersPacific= FOREACH filterUsersPacific GENERATE
userId,userName,friendCount,followerCount,statusCount,favoriteCount,state,timezone;

projectedNetworks= FOREACH networks GENERATE followerId;
projectedTweets= FOREACH tweets GENERATE tuserId,originalText,retweetCount,hashTags;

-- Join the projected user profile dataset with the Networks table on userId to get the records of only the followers
joinUsersCentralAndNetworks= JOIN projectedUsersCentral BY (userId), projectedNetworks BY (followerId);
-- Join the previously joined results with the projected tweets table to get tweet information of the followers
joinFollowersAndTweetsCentral= JOIN joinUsersCentralAndNetworks BY (userId), projectedTweets by (tuserId);

joinUsersEastAndNetworks= JOIN projectedUsersEast BY (userId), projectedNetworks BY (followerId);
joinFollowersAndTweetsEast= JOIN joinUsersEastAndNetworks BY (userId), projectedTweets by (tuserId);

joinUsersMountainAndNetworks= JOIN projectedUsersMountain BY (userId), projectedNetworks BY (followerId);
joinFollowersAndTweetsMountain= JOIN joinUsersMountainAndNetworks BY (userId), projectedTweets by (tuserId);

joinUsersPacificAndNetworks= JOIN projectedUsersPacific BY (userId), projectedNetworks BY (followerId);
joinFollowersAndTweetsPacific= JOIN joinUsersPacificAndNetworks BY (userId), projectedTweets by (tuserId);

-- Group each joined table by state for each time zone
groupDataCentral= GROUP joinFollowersAndTweetsCentral BY (state);
groupDataEast= GROUP joinFollowersAndTweetsEast BY (state);
groupDataMountain= GROUP joinFollowersAndTweetsMountain BY (state);
groupDataPacific= GROUP joinFollowersAndTweetsPacific BY (state);

-- Determine the character of the followers in each time zone by applying aggregate functions
```

```
characterCentral= FOREACH groupDataCentral GENERATE group, MAX($1.$11) AS
maximumRetweets,AVG($1.$2),AVG($1.$3), AVG($1.$4), AVG($1.$5);
characterEast= FOREACH groupDataEast GENERATE group, MAX($1.$11) AS
maximumRetweets,AVG($1.$2),AVG($1.$3), AVG($1.$4), AVG($1.$5);
characterMountain= FOREACH groupDataMountain GENERATE group, MAX($1.$11) AS
maximumRetweets,AVG($1.$2),AVG($1.$3), AVG($1.$4), AVG($1.$5);
characterPacific= FOREACH groupDataPacific GENERATE group, MAX($1.$11) AS
maximumRetweets,AVG($1.$2),AVG($1.$3), AVG($1.$4), AVG($1.$5);

-- Store the results for each time zone
STORE characterCentral INTO 's3://mrprojectsarika/outputFPJ/central' USING PigStorage(',');
STORE characterEast INTO 's3://mrprojectsarika/outputFPJ/east' USING PigStorage(',');
STORE characterMountain INTO 's3://mrprojectsarika/outputFPJ/mountain' USING PigStorage(',');
STORE characterPacific INTO 's3://mrprojectsarika/outputFPJ/pacific' USING PigStorage(',');
```

Execution Plans:

Execution Plan 1: Join Order

This execution plan works on changing the join Order and trying to determine the effect of this change.While execution Pig works in such a way that the last table is not brought into memory but streamed through instead. Hence the memory usage is better if we place the last table to the bigger table. Hence this should improve the performance by reducing the memory usage and records spilled.

Execution Plan 2: Switch Off the MultiQuery Optimizer

By default the MultiQuery optimizer is switched on. This optimizer parses the script once before executing it and determines the steps that it can combine. By doing so it reduces the steps to be executed and hence reduces the running time.

Execution Plan 3: Use of Parallel clause

Using Parallel clause with statements that begin the reducer, will help parallelize the reduce phase, which in turn will reduce the number of reduce tasks.This should decrease the running time of the reducer and in turn the running time of the pig script.

Execution Plan 4: Switch off Push Up Filter

Push Up Filter is another optimization that is applied by Pig to push up the FILTER operators up the data flow graph. This decreases the amount of data that flows through the pipeline. By switching this feature off we are expecting the data spilled in memory to be more, which would be comparatively less in the otherwise situation.

Execution Plan 5: Varying the order in which we perform Selection, Projection and Join

In general early Projection improves the performance of the Pig scripts as we discard the attributes which we would not require in the execution of the scripts. This reduces the amount of data transfer as well as aids in decreasing the processing time. Hence we expect joinPFJ and joinPJF to have less data tranfser then the other approaches. In terms of running time we expect the joinJPJ and joinJFP to be faster as we join first and then the filter and projection opporations will be pushed up because of the PushUpFilter optimizers. This is decrease the processing time of the script.

We ran the Pig scripts in the following configuration

- 1 x m1 large machine - master
- 10 x m1 large machines - worker

Technical discussions and Comparisons:

Execution Plan	Number of Maps	Number of Reduce	Running time
joinOrder	204	30	17min 17sec
joinParallel	218	50	17min 13sec
joinSwitchOffPushUpFilter	204	30	18min
joinFJP	349	120	32min 53sec
joinFPJ	560	120	34min 33sec
joinJFP	190	30	19min 27sec
joinJPF	204	30	17min 11sec
joinPFJ	560	120	34min 13sec
joinPJF	366	30	17min 42sec

<u>Analysis:</u>

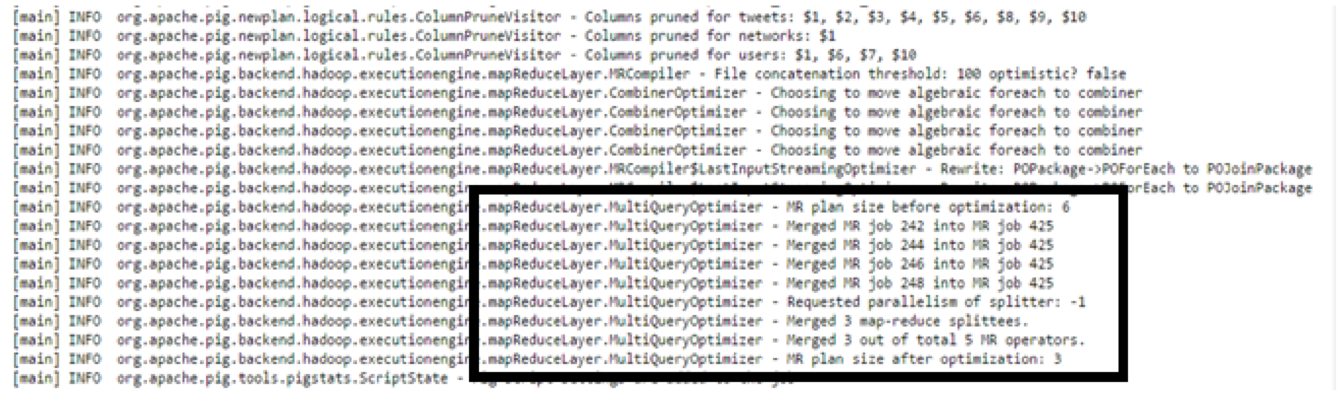
MultiQuery Optimizer:

We tried switching off the MultiQueryOptimizer by running the joinOrder script with the following command

pig -M joinOrder.pig

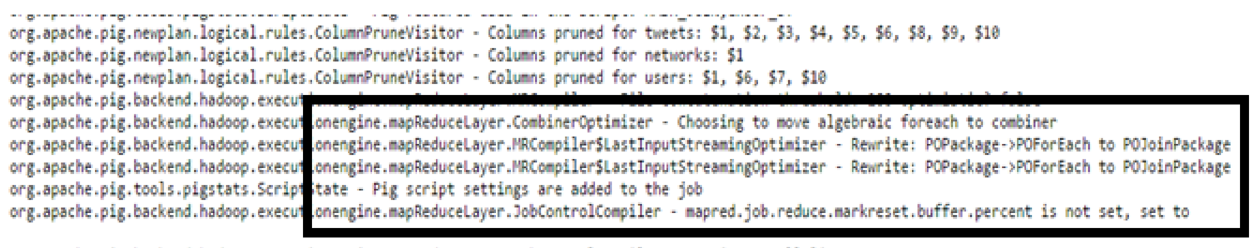
The argument -M would switch off the MultiQuery Optimizer and hence it is not applied in the script which can be visible in the below screenshot.

joinOrder with Optimizers turned On: [refer : stderr_joinOrder.txt]



```
[main] INFO  org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for tweets: $1, $2, $3, $4, $5, $6, $8, $9, $10
[main] INFO  org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for networks: $1
[main] INFO  org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for users: $1, $6, $7, $10
[main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false
[main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.CombinerOptimizer - Choosing to move algebraic foreach to combiner
[main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.CombinerOptimizer - Choosing to move algebraic foreach to combiner
[main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.CombinerOptimizer - Choosing to move algebraic foreach to combiner
[main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.CombinerOptimizer - Choosing to move algebraic foreach to combiner
[main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler$LastInputStreamingOptimizer - Rewrite: POPackage->POForEach to POJoinPackage
[main] INFO  org.apache.pig.backend.hadoop.executionengin                                                   orEach to POJoinPackage
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 6
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Merged MR job 242 into MR job 425
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Merged MR job 244 into MR job 425
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Merged MR job 246 into MR job 425
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Merged MR job 248 into MR job 425
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Requested parallelism of splitter: -1
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Merged 3 map-reduce splittees.
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - Merged 3 out of total 5 MR operators.
[main] INFO  org.apache.pig.backend.hadoop.executionengin .mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 3
[main] INFO  org.apache.pig.tools.pigstats.ScriptState -
```

joinOrder with MultiQueryOptimizers turned Off: [refer : stderr_joinOrderSwitchOffMultiQuery.txt]



```
org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for tweets: $1, $2, $3, $4, $5, $6, $8, $9, $10
org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for networks: $1
org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for users: $1, $6, $7, $10
org.apache.pig.backend.hadoop.execut[illegible]
org.apache.pig.backend.hadoop.execut onengine.mapReduceLayer.CombinerOptimizer - Choosing to move algebraic foreach to combiner
org.apache.pig.backend.hadoop.execut onengine.mapReduceLayer.MRCompiler$LastInputStreamingOptimizer - Rewrite: POPackage->POForEach to POJoinPackage
org.apache.pig.backend.hadoop.execut onengine.mapReduceLayer.MRCompiler$LastInputStreamingOptimizer - Rewrite: POPackage->POForEach to POJoinPackage
org.apache.pig.tools.pigstats.Script tate - Pig script settings are added to the job
org.apache.pig.backend.hadoop.execut onengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markreset.buffer.percent is not set, set to
```

As you can see in joinOrder.pig script, the MultiQueryOptimizer has reduces the MR plan size from 6 to 3, that is by 50%. But when we switched off the optimization, it affected the running time a lot. It increased from 17 minutes to **59 minutes.**
Since the scripts were not combined, but executed separately for each STORE statement it increased the total number of maps and reduce tasks. Since there are 4 STORE statements, for each store it executes the map and reduce tasks.

Pig Script	Number of Maps	Number of Reduce	Running time
joinOrderSwitchOffMultiQuery	784	120	59 minutes

Switching Join Order:
I compared 2 scripts , first script joinSplit.pig has the last table in the join to be the table with smaller number of tuples per key and the second script joinOrder.pig has the table with larger number of tuples per key to be the last table in the join command.

Comparision of performance:[refer docs stderr_joinSplit.txt and stderr_joinOrder.txt]

Pig Script	Number of Maps	Number of Reduce	Running time
joinSplit	218	50	17 min 44 sec
joinOrder	204	30	17 min 17min

As shown above even though the running time is not affected a lot, the average time to perform Reduce tasks has increased a lot.

Use of Parallel Clause:
We set the parallel clause to 20 in the Join statement as it is one of the commands that begins the reduce phase. We were hoping that this would affect the performance of the script but this did not affect the performance or the running time of the pig script. the running time of the script ended up being almost the same as it was before setting the Parallel clause.

PushUpFilter:
This optimization is turned on by default. We executed the script joinJPF.pig using the following command on AWS

```
pig -t PushUpFilter joinJPF.pig
```

Pig Script	Number of Maps	Number of Reduce	Running time
joinJPF	204	30	17 min 11 sec

joinJPF_SwitchOffFlter	204	30	18 min

As you can see from the above table, switching off the filter has slightly affected the running time, but this is not a big difference. The reason was that the filter statements were getting optimized by the MultiQuery optimizer and hence there was optimization being applied to the script and so the performance did not get affected.

Column Pruning:
We can notice from the stderr file that only the required attributes from each data set is pruned and passed on only the necessary columns. This pruning of columns improves the efficiency of the program.

2015-04-22 13:25:30,336 [main] INFO org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for tweets: $1, $2, $3, $4, $5, $6, $8, $9, $10
2015-04-22 13:25:30,340 [main] INFO org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for networks: $1
2015-04-22 13:25:30,341 [main] INFO org.apache.pig.newplan.logical.rules.ColumnPruneVisitor - Columns pruned for users: $1, $6, $7, $10

Pig performs this column pruning automatically. We can observe this pruning applied in all stderr.txt file for all Pig scripts.

Early Projection and Selection:
Of the various Projection, selection and join order , the script with Projection, Join and Selection order has the best performance in terms of running time. This script has worked significantly better than Projection, Selection and Join because the number of records after joining has become significantly less than before the join. it is also better than performing Filter first for the same reason.

Conclusion:
Take aways from executing the various execution plans are

- Even though early and often projection and selection are good ways to optimize the script, since the results we got after joining tables were small, performing joining in the beginning followed by Projection and selection seemed to be the optimized way of executing this particular script.
- We were hoping than when I switch off the PushUpFilter the performance would get affected. But after taking a careful look at the stderr files, we relaized that the MultiQuery optimizer was applied in the beginnng itself for the 4 filter queries I had use to filter on longitude.
- Finally we relaized the impact of turning MultiQuery optimizer when I turned it off and ran the code. Each STORE command's relate queries ran separately which led to reading the datasets 4 times, applying selection and projection 4 times. This resulted in a running time which was more than double of the normal running time.

[References : http://chimera.labs.oreilly.com/books/1234000001811/ch07.html#explain
https://pig.apache.org/docs/r0.9.1/perf.html#filter
http://pig.apache.org/docs/r0.8.1/piglatin_ref1.html
https://pig.apache.org/docs/r0.11.1/perf.html#SplitFilter]