**Problem**

*Twitter data questions, #3:* How does @PrezOno’s tweet length compare to the average of all others? What is his average length? All others?

**Approach**

In this problem, we needed to read through Twitter JSON data and grab each tweet’s ‘text’ and user ‘screen\_name’ values. The number of characters (length) for each tweet would be calculated using the ‘text’ value, and the ‘screen\_name’ would be used to find out if a tweet was from @PrezOno. At first we attempted to run the mapreduce programs over the entire /data/twitter/ directory, but that proved to be problematic as the jobs were very slow and ultimately unsuccessful. We decided to limit the dataset to use only tweets from December, but this still ran fairly slow. There was a lot of extra data that we didn’t need in the tweets, so to increase the speed of mapreduce jobs we decided it was necessary to implement a pre-mapreduce map function to read all of the JSON data and output only the data we would use, as plain text. The data outputted was “screen\_name, tweet\_length, tweet\_date, popularity, tweet\_id” all separated by tabs (tweet\_date, popularity, and tweet\_id are used in problem #10 and not implemented in this situation). The tweet\_length was calculated by using Python’s “len()” function on the value of the ‘text’ key (i.e. *tweet\_length =*  *len(tweet[‘text’])* ).



*Sample set of data used as the input for the mapreduce program.*

*In the format “screen\_name, tweet\_length, tweet\_date, popularity, tweet\_id”*

**Map**

In the map function of the program, we created global variables to contain the sum of PrezOno’s tweet lengths, the number of PrezOno’s tweets, the sum of all other users’ tweet lengths, and the number of tweets from all other users. As we iterated through each line of the input, we parsed the current line by tabs and stored the screen\_name and tweet\_length values. If the screen\_name was “PrezOno” then we would add the tweet\_length to the running sum of PrezOno’s tweet lengths, and add 1 to the running count of PrezOno’s tweets. If the screen\_name was not “PrezOno”, then we would add the tweet\_length to the running sum of all other users’ tweet lengths, and add 1 to the running count of all other users’ tweets. Once the entire input was read, we printed “PrezOno”, PrezOno’s total tweet\_length sum, and PrezOno’s total tweet count, with each value separated by a tab. We also printed “OtherUser”, the total sum of tweet\_lengths from every user excluding PrezOno, and the total tweet count from every user excluding PrezOno, with each value separated by a tab.

**Reduce**

The reduce function was very similar to the map function, in that we are creating global variables to contain the sum of PrezOno’s tweet lengths, the number of PrezOno’s tweets, the sum of all other users’ tweet lengths, and the number of tweets from all other users. The data printed by the map function was used as input for the reduce function, which we iterated through and parsed by tabs, storing the key (i.e. “PrezOno” or “OtherUser”), length sum, and tweet count into variables. The key value was then checked to see if it was “PrezOno”, and if it was, the current length sum value was added to the global PrezOno length sum and the current tweet count was added to the global PrezOno tweet count. If the key was “OtherUser”, then the current length sum value was added to the global “all-other-users” length sum and the current tweet count was added to the global “all-other-users” tweet count. Once we read through the entire input, if there were no “PrezOno” tweets then we printed “No tweets from PrezOno”. If there were “PrezOno” tweets, then we divided the global sum of PrezOno tweet lengths by the global number of PrezOno tweets and printed the result, which is the average length of PrezOno’s tweets. We also divided the global sum of tweet lengths by the global number of tweets, for all other users, and then printed the result, which is the average tweet length of all users excluding PrezOno.



*Output of the reduce function*

**We found that for the month of December @PrezOno’s average tweet length was about 15 characters more than the average tweet length of all other users.**

**Problem**

*Twitter data questions, #10:* For each day, what was the most retweeted or most favorited tweet?

**Approach**

In this problem, we needed to read through Twitter JSON data and grab each tweet’s ‘id’, ‘created\_at’, ‘retweeted\_count’, and ‘favorited\_count’ values. Since we had to find the most retweeted *or* most favorited tweet, we interpreted that as using the larger of the two values (retweeted\_count and favorited\_count) as the deciding factor. The larger value would then be referred to as the “popularity”, and every tweet’s popularity value would be compared with each other within a specific day, in order to find the most popular (retweeted/favorited) tweet for that day. We also decided to use the tweet\_id when referring to tweets instead of the tweet text, because it is much more readable, less data, the jobs would run faster, and we wouldn’t need to consider encoding issues when dealing with special characters in text. At first we attempted to run the mapreduce programs over the entire /data/twitter/ directory, but that proved to be problematic as the jobs were very slow and ultimately unsuccessful. We decided to limit the dataset to use only tweets from December, but this still ran fairly slow. There was a lot of extra data that we didn’t need in the tweets, so to increase the speed of mapreduce jobs we decided it was necessary to implement a pre-mapreduce map function to read all of the JSON data and output only the data we would use, as plain text. The data outputted was “screen\_name, tweet\_length, tweet\_date, popularity, tweet\_id” all separated by tabs (screen\_name and tweet\_length are used in problem #3 and not implemented in this situation). The popularity was calculated using Python’s “max()” function on the ‘retweeted\_count’ and ‘favorited\_count’ values (i.e. *popularity = max(retweeted\_count, favorited\_count)* ).



*Sample set of data used as the input for the mapreduce program.*

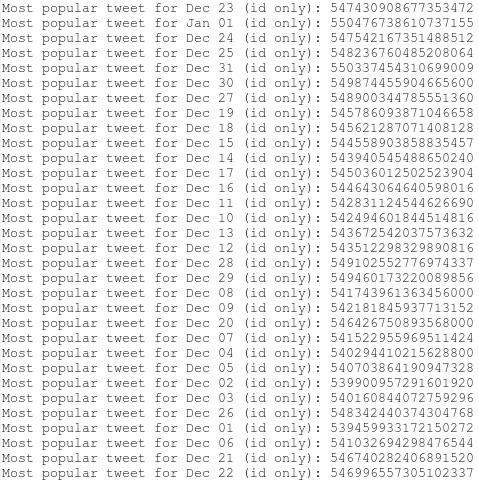
*In the format “screen\_name, tweet\_length, tweet\_date, popularity, tweet\_id”*

**Map**

When reading through the data, we needed some way to store each date, its most popular tweet, and the popularity value of the tweet, and then refer back to that day’s current most popular tweet if we encountered another tweet from the same day, to compare if the new tweet is more popular. To solve this we created a global dictionary object to store a date as the key, with the value being the tweet\_id and tweet’s popularity value. As we iterated through each line of the input, we parsed the current line by tabs and stored the tweet\_date, tweet\_id, and popularity. If the current tweet\_day wasn’t a key in the global day-dictionary ***or*** if the current tweet’s popularity was greater than the popularity stored in the day-dictionary for a particular day, create/edit an entry in the global day-dictionary where the key is the currently read tweet’s day with a value of the tweet’s popularity and tweet\_id. Once the entire input was read, we iterated through the global day-dictionary, printing the keys (days), popularity, and tweet\_id, separated by a tab.

**Reduce**

The reduce function was almost exactly the same as the map function, because if there are multiple entries for a specific day resulting from the different map functions that were run, then we would again need to compare the popularity for each day. We created a global day-dictionary variable again, then iterated through the input and parsed it by tab, storing the key (date), popularity, and tweet\_id into variables. If the current tweet\_day wasn’t a key in the global day-dictionary ***or*** if the current tweet’s popularity was greater than the popularity stored in the day-dictionary for a particular day, create/edit an entry in the global day-dictionary where the key is the currently read tweet’s day with a value of the tweet’s popularity and tweet\_id. When we were finished reading the input, we iterated through the global day-dictionary and printed the keys (day) with their values, i.e. popularity and tweet\_id. The result was a list of each day in December’s most popular (retweeted or favorited) tweet.



*Output of the reduce function*

**Problem Definition:**

Twitter Problem #2: What day of the week does @PrezOno tweet the most on average? Use the same example as in #1 but for days of the week.

In this problem we are calculating which day of the week President Ono tweets the most on average across all available Twitter data on the Hadoop cluster.

**Solution:**

To solve this we wanted to look through all tweets in the Twitter dataset and record what days PrezOno tweeted as well as a total count.

Map:

After reading in each tweet as JSON, we would look for the ‘user’ attribute followed by the ‘id’ attribute. If that id matched President Ono’s (211178363) then we would print out a key/value pair. This key value pair consisted of the key ‘1’ and the value of the first 3 characters of the tweet data’s ‘created\_at’ attribute. This attribute always began with a three letter abbreviation for the day the tweet was created (i.e. Mon, Tue, Wed).

Reduce:

The reduce for this function contained a count for Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, and an overall count for all 7 days. We went through all the printed out key values and would increment the overall count as well as the associated day’s count (known from the value of the key/value pair). After aggregating the counts we calculated the average for each day by dividing the individual day’s count by the total count. These averages gave us our overall percentage for each day.

**Results:**

We found that President Ono tweets the most on Thursday (~15.7%). The results were much more closely grouped than we thought they would be, indicating that he does tweets regularly during the whole week.

The final numbers are reported below.

|  |  |
| --- | --- |
| **Mon** | **11.8%** |
| **Tue** | **13.6%** |
| **Wed** | **15.4%** |
| **Thu** | **15.7%** |
| **Fri** | **15%** |
| **Sat** | **13.7%** |
| **Sun** | **14.9%** |

**Problem Definition:**

Twitter Problem #5: What twitter user tweeted the most? What is the top 5 longest tweeters? Bottom 5?

In this problem we are reporting which user tweeted the most individual tweets. We are also reporting which 5 users tweeted the most characters overall and which 5 users tweeted the least characters overall. This problem is only across the dataset for December 2014 as Hadoop cluster issues interfered with our runs over the full dataset.

**Solution:**

To solve this we went through all tweets in the dataset and kept a count of how many times each individual user tweeted and their total character count over all recorded tweets.

Map:

We kept a dictionary that contained each user as a key and a list of their total tweets and total character count as the value. To make this dictionary we found the ‘user’ attribute of each tweet followed by the ‘screen\_name’ attribute. This was how we identified each user. We then recorded the length of the ‘text’ attribute of each tweet. This information (identifier and character count) were put into a list.

After all the data was recorded into a list we iterated through each list entry and made a new dictionary entry for any unique user consisting of their name as the key and the value 1 for total tweets and the value pulled from the list for their character count. If a user was already present in the dictionary we incremented their tweet count by one and added the character count from the list to the total character count in the dictionary entry.

After all data was put into the dictionary we wrote the information out to a list, sorted by the total character count entry in the dictionary. We then printed the top and bottom five users from the dictionary as their name, their total tweets, and their total tweet lengths.

Reduce:

The reduce kept two dictionaries, one for keeping track of total tweets and another for total tweet length. For each user we would check for their existence in each dictionary and increment the values accordingly (add the value to the preexisting value in the dictionary). If a user was new we added them to the dictionary with their initial values as the values pulled from the map step.

After putting the dictionaries together we sorted the two dictionaries and printed the top and bottom 5 entries for each dictionary. This gave us our total value for who tweeted the most/least individual tweets and who tweeted the most/least individual characters.

**Results:**

We found that the top tweeter was DogTreatNews with a total of 12,644 individual tweets.

The users with the most/least tweeted characters are listed below (with charts to help visualize).

|  |  |
| --- | --- |
| Longest Tweeters | |
| DogTreatNews | 1659095 |
| OhioHealthJobs | 413276 |
| BrittanyT820 | 373242 |
| 2dayswork | 338479 |
| Idk\_Caniff | 303365 |

|  |  |
| --- | --- |
| Shortest Tweeters | |
| Mariahsdavis | 1 |
| Kmay330 | 1 |
| Cam2cocky | 1 |
| Alynikoll | 1 |
| Tanna\_stables | 1 |

**Problem Definition:**

Twitter Problem #1: What hour of the day does @PrezOno’s tweet the most on average, using every day we have twitter data?  Include a plot of the expected number of tweets for each hour of the day, for those he did tweet.  For example if Ono tweeted once every day at 12:30PM, his expected number of tweets between 12 and 1 would be 1.  If he alternates between 2 and 3 tweets per day, his average would be 2.5.

In this problem, we are finding the average number of tweets that President Ono (@prezOno) tweets for every hour of the day.

**Solution:**

In order to solve this, we search through all of President Ono’s tweets, find out what hours they were tweeted at, and keep track of the total tweets that President Ono tweeted.

Map:

The initial step of this problem is to parse the JSON, and determine if it was a tweet by President Ono. To do this, pull the ‘user’ attribute, then either the ‘id’ or ‘display\_name’ attributes from the resulting dict and compare them to President Ono’s account’s value (id=211178363 display\_name=”prezOno”). If it matches, pull the hour (found in 00-23 hour format) from the ‘created\_at’ attribute of the tweet, and print that value as the key, and 1 as the value.

Reduce:

The reduce portion of this task is fairly simple. I started by creating a dictionary, where each key was an hour of the day (00-23) and the value was initialized to 0 as well as a ‘total’ value. This makes the loop very simple, as all that needs done is to split the key/value pair, then increment the dictionary at the key by the value and increment the total by the value. After the loop portion, simply loop through the dictionary, and print the value divided by the total. If done correctly, this would give each hourly average. Incrementing by the value leaves room for optimization in the map function.

**Results:**

We performed our calculations against the entire provided Twitter dataset on the Hadoop cluster, and found that President Ono tweets 6.98% of the time between 02 and 03 GMT, or 22 and 23 EST. President Ono’s tweet count is fairly consistent throughout the day, but makes a clear drop between the hours of 05 and 10 GMT (01 and 06 EST).

**Problem Definition:**

Twitter Problem #6: For each day of the week, what was the most mentioned hashtag?  Hour of the day?

This problem has two parts: 1) Calculating the most mentioned hashtag in relation to the day of the week and 2) calculating the most mentioned hashtag in relation to the hour of the day.

**Solution:**

In order to solve this problem, we must count all of the hashtags for each hour, as well as each day of the week, then find the maximum value for each of these.

Map:

The map for this problem is very simple. For every tweet, pull out the day and the hour, like in problems 1 and 2, and pull out the list of hashtags. To get the hashtags, access the entities attribute, then the hashtags attribute. Loop through the list of hashtags, and for each hashtag print the hashtag’s text attribute, the day of the week, and the hour of the day. Note, this problem requires a key/value/value set, as opposed to the normal key/value pair.

Reduce:

The reduce step for this problem was a bit tricky. The function needs to keep track of each instance of different tweets across all days of the week and hours of the day. The way we tackle this is to create two dictionaries of dictionaries. The top-level dictionaries will have the days of the week and hours of the day as keys, and second-level dictionaries as values. The second-level dictionaries would contain the hashtag text as a key, and the number of occurrences on that day as the value. For each hashtag passed through, we either add it to the appropriate second level dictionary with a start value of 1, or increment an existing second-level dictionary value by 1. After the primary loop, we calculate the most-occurring hashtag for each key in the first-level dictionaries, and print those out.

**Results:**

This was a very heavy job, and we ended up running it against all tweets within the month of June in 2014. We found that the most mentioned hashtag daily was #winning on Wednesdays at 31852 instances, and the most mentioned hashtag hourly was #dog at 23 GMT at 77975 instances.