Project Report: Finding the Most Popular Twitter Hashtags in a Region

Group Members: Palak Parwal, Michelle Wang, Jay Song

Requirements

The overall goal of the this project is to collect 5 Gigabytes of tweets and use the location and hashtags from the tweets to find the most popular tweets in a region. This goal can be broken down into specific requirements below.

1. Obtain 5 Gigabytes of data from a social network
    1.1. A crawler that will request a stream of tweets from the Twitter Streaming API
        1.1.1. There must be a crawler with the proper authentication
    1.2. Data should be saved to a JSON file for storage until it is needed next
2. Use Hadoop to MapReduce the raw data we collected
    2.1. Use the Map portion to create (location, hashtag) pairs for each tweet
    2.2. Organize these pairs by location.
    2.3. Reduce these organized pairs to grab the most popular hashtag based on location.
    2.4. Stored pairs into Cassandra to be used in the last portion.
3. Use the Google Maps JavaScript API to map out what hashtag is most popular in what location
    3.1. Use Ruby on Rails as a web framework to design the website with a Model-View-Controller setup
    3.2. Connect Apache Cassandra to Rails so the controller can communicate with the database
        3.2.1. The results from queries are output to the website in a view
    3.3. Use Rails to locally host a development server for the website

Design

To gather the information we had to gain access to the Twitter Streaming API.

After that we had to give the stream the specific filters we would like for it to apply to the incoming stream of tweets. For our purposes, this included specifying that we wanted the tweets to be geolocated. We gave it a specified box that would include the United States of America mainland. It also includes bits of Canada and Mexico as the box could not be shaped.

Next, we wrote each incoming tweet to a file for storage. We noticed that not every line had a tweet on it. To get rid of these lines, we checked for a text field on every line and removed any lines that did not contain one. While doing this, we also split the remaining data (after removal of the lines without tweets) into files that were approximately 100 Megabytes each.

For the next portion, we wrote two scripts in python that would map and reduce this cleaned data. We were able to do this thanks to the Hadoop Streaming jar that made it so we didn't have to use Java.

The Mapper takes in the directory with the cleaned data files and checks to see if a tweet has hashtags and has a full_name for the place attribute. This ensures that the tweets we look at have all of the information we need. It removes the tweets that will not be useful. If the tweet meets these conditions, the Mapper outputs pairs of location and hashtag. It will output one pair for every hashtag that appears.

Hadoop then shuffles these pairs to be sorted by location. This is because it sorts by the key of each pair before giving the pairs to the Reducer.

After this, the shuffled output of the Mapper is sent to the Reducer. The Reducer creates a dictionary to count the number of times a hashtag appears for a location. When it sees a new location, it outputs the current location's most popular hashtag based on the count and resets the dictionary. This output is compiled into a file in the format of:
Location\tHashtag\tCount

Since we did this process daily, we were able to track which day we acquired the tweets. We used this information to append the date to each line in the file, from the MapReduce portion. The new file format is:
Location\tHashtag\tCount\tDate

The file of this format is what we used next to populate a table in Cassandra. The same script that appends the dates also writes a script to populate the Cassandra database with this information. It then populates the table with the information in the appended file with the location and date being the Primary Key as a pair.

We also wrote a second mapper in python that takes in the directory with the cleaned data files and checks to see if a tweet has hashtags, a full_name for the place attribute, and has coordinates. This ensures that the tweets we look at have all of the information we need. It removes the tweets that will not be useful. If the tweet meets these conditions, the Mapper outputs the hashtag, location, latitude, and longitude. It will output one set for every hashtag that appears. The file that is output is of the form:
Hashtag\tLocation\tLatitude\tLongitude

This data is used by the same script that populates one table of our databased to populate another. The output from the second Mapper has a date and an ID appended to each line. The new format becomes:
Hashtag\tLocation\tLatitude\tLongitude\tID\tDate

The file of this format is what we used next to populate another table in Cassandra. The same script that appends the dates and IDs also writes a script to populate the Cassandra database with this information. It then populates the table with the information in the appended file with the id, date, hashtag, and location being the Primary Key as a set. This process is done daily as well with the new data that comes in.

The next portion of the project requires us to display the data on a website. In order to do this, we used Ruby on Rails. We connect it to the Cassandra database using a Cassandra driver for Ruby and place the hashtags on a map using the Google Maps API. In specific, we used the Google Maps JavaScript API to display the data.

The website will take a location, a location and date, or a hashtag and date as input options from the user. If only a location is given, the past 5 days worth of most popular hashtags will be output above the location. If a date is specified as well, and the date is one of the dates in our database, only the hashtag for that date will be output and there will be red dots all over the area to show where the hashtags were sent from. This will only include the hashtags for which we have coordinate values. If a hashtag and date are specified, then a heatmap of that hashtag on that date will be shown over the map. If the input is invalid, a pop up will inform the user of proper usage methods.

Implementation

The crawler, twitter_streaming.py, works as follows in python using the tweepy library:

1. It has been manually given the Consumer Key, Consumer Secret Key, Access Token, and Access Token Secret
2. It used these values to get an authentication token through the tweepy library's OAuthHandler
3. It then give the stream a filter in the form of a pair of coordinates
    3.1. These coordinated are the southwest and northeast corners of the rectangle we made around the United States
4. The stream is then started
5. Each tweet is received and written to the opened file
6. The file is closed each time to prevent data loss

The cleanup script, dataresize.py, works as follows:

1. It takes in the data file

2. It goes through and looks for lines that contain a text field
    2.1. These lines are rewritten to a new file
3. The lines are written to a file until the file's size is greater than or equal to 100 Megabytes
    3.1. The number of files is based on the number required to hold the raw data in around 100 Megabyte increments

The Mapper, mapper.py, works as follows:

1. It takes in the directory of separated data files
2. It checks to make sure the tweet it is looking has contains a hashtag as well as the full_name value in the place attribute
3. If the tweet meets the requirements, the Mapper outputs pairs of the form location\thashtag for each hashtag that appears within the tweet
    3.1. The key is the location
    3.2. The value is the hashtag

Hadoop then shuffles these pairs based on the key

The Reducer, reducer.py, works as follows:

1. It creates a dictionary that will help figure out which hashtag is the most popular
    1.1. It uses the hashtag as the key and the value is a counter
2. It creates another dictionary that will track each location's most popular hashtag and the number of times it appears
3. It takes in the output of the Mapper one pair at a time
4. While the location is the same
    4.1. It will add each new hashtag to the count dictionary and set the value to 1
    4.2. If the hashtag is not new, it will increment the value in the dictionary
5. When it gets a new location
    5.1. It will save the current location, most popular hashtag in the count dictionary for that location, and the number of times that hashtag appeared at that location to a location dictionary
    5.2. It will then continue on to the next location's values
6. In the end it outputs the location dictionary in the format: location\thashtag\tcount

The second Mapper, mapper2.py, works as follows:

1. It takes in the directory of separated data files
2. It checks to make sure the tweet it is looking has contains a hashtag, a full_name in the place attribute as well as a coordinates
3. If the tweet meets the requirements, the Mapper outputs information of the form: hashtag\tlocation\tlatitude\tlongitude for each hashtag that appears within the tweet
4. We do not reduce this because we want all of this information

The script to run Hadoop, endtwitter.py, works as follows:

1. It stops the collection of tweets that is started by the twitter_streaming.py script

2. It then calls the dataresize.py script to remove lines that are not tweets and parse the data into files of about 100 Megabytes each
3. It then sends the directory of these files to Hadoop and runs the first MapReduce set
    3.1. The output from this is sent to a new location and the output file is deleted from Hadoop
4. The directory is then sent through the second Mapper to get the other set of data
    4.1. This output is also sent to a new location before the output file is deleted from Hadoop

The script that imports the values into Cassandra tables, importdailydata.sh, works as follows:

1. For the first table:
    1.1. It appends the date on which the values were acquired to each line in the file from the Reducer so that the new format is: location\thashtag\tcount\tdate
    1.2. It then creates a script that will add these values to the database as follows:
        1.2.1. It will create the Keyspace cs179g if it does not already exist
        1.2.2. It will then create the table mytable if it does not exist
        1.2.3. The values in the table are location (text), hashtag (text), count (int), and date (text)
        1.2.4. The Primary Keys are location and date as a pair
    1.3. The values from the new appended file are then fed to this script to add this new data to the database
2. For the second table
    2.1. It appends the line number as an ID and the date on which the values were acquired to each line in the file from the second Mapper so that the new format is: hashtag\tlocation\tlatitude\tlongitude\tid\tdate
    2.2. It then creates a script that will add these values to the database as follows:
        2.2.1. It will create the Keyspace cs179g if it does not already exist
        2.2.2. It will then create the table htcrd if it does not exist
        2.2.3. The values in the table are hashtag (text), location (text), latitude (text), longitude (text), id (int), and date (text)
        2.2.4. The Primary Keys are id, date, hashtag, and location as a set
    2.3. The values from the new appended file are then fed to this script to add this new data to the database

The Ruby on Rails application works as follows:

1. The routes.rb file is used to route the requests it receives to the correct controller
2. The pages_controller.rb file takes in what the routes.rb file sends and process the request
    2.1. It will respond to the information by performing the action and returning the proper view
        2.1.1. The home action of the pages_controller will return the home view which is just the map and the input boxes

2.1.2. The search action of the pages_controller will do a search for most popular hashtag with or without the date
2.1.2.1. It will return the home view if the search was not valid or had no results
2.1.2.2. It will return the search view if the search was valid and returned results
2.1.3. The heatmap action of the pages_controller will do a search for a hashtag on a specified day
2.1.3.1. It will return the home view if the search was not valid or had no results
2.1.3.2. It will return the search view if the search was valid and returned results

3. The views file will take in the results from the pages_controller.rb and generate an html file based on what it receives
3.1. The home.html.erb file will generate the map with the input boxes
3.1.1. It can also have a pop up alert if the last set of values inputted was invalid
3.1.1.1. The message is based on the error
3.2. The heatmap.html.erb file will generate a heatmap based on the results of the search and output a JavaScript portion for an HTML file that will generate the map using the Google Maps API
3.2.1. When giving the latitude and longitude values to the API, we had to switch them as they were incorrectly labeled in the database
3.3. The search.html.erb file will use the results to generate the JavaScript portion of an HTML file that will generate a map using the Google Maps API
3.3.1. This map will include a marker over the city name with the hashtag and count information as well as smaller markers for the hashtags with coordinate values

Evaluation

The original data collection took a total of approximately 13 hours and 30 minutes. In this time we collected 5.2 Gigabytes of data. We used the school's well server to speed up the data collection process. It cut the time in half from what we had originally estimated.

This data contains the text of the tweets, the location, and any other information Twitter deemed important for the tweets. The data is held in a JSON file because that is how Twitter sends the information. The 5.2 Gigabytes contain 1,734,435 lines of data.

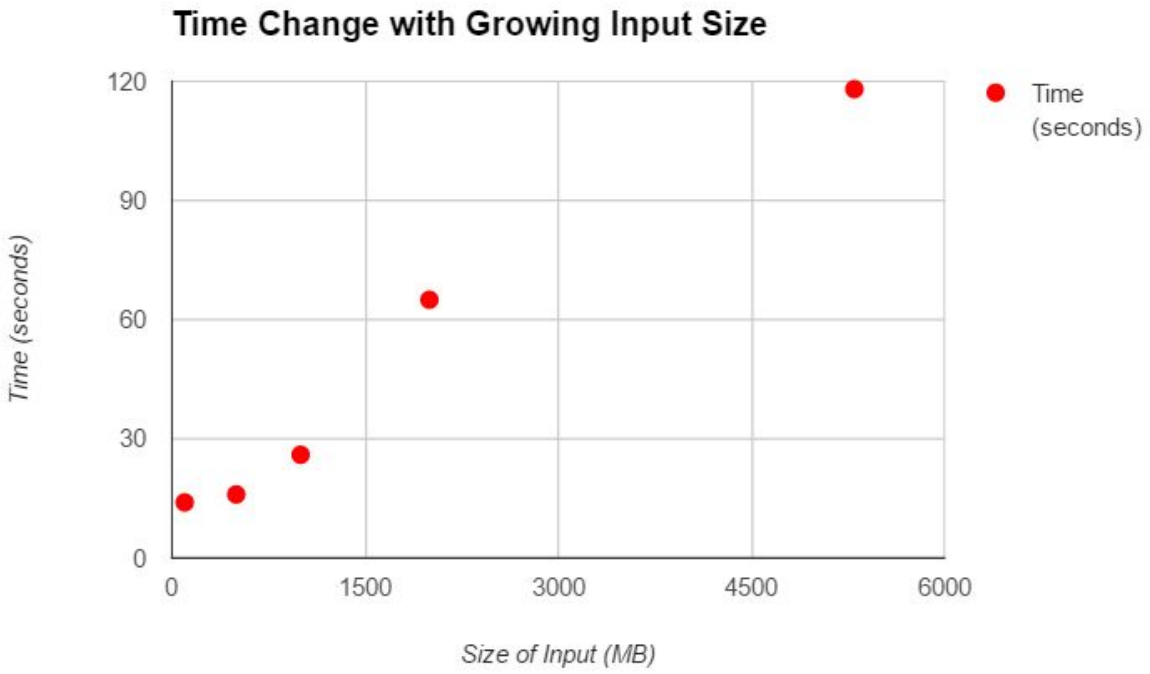
After cleaning up the data to remove lines that were not tweets, our raw data is of size 5.1 Gigabytes. This means only 100 Megabytes of the data was invaluable. The total number of lines of data after cleaning up was 1,693,666.

This means there are 1,693,666 tweets and the remainder from the original line count were the lines that did not contain a tweet. There were 40,769 lines of non tweets in the file.

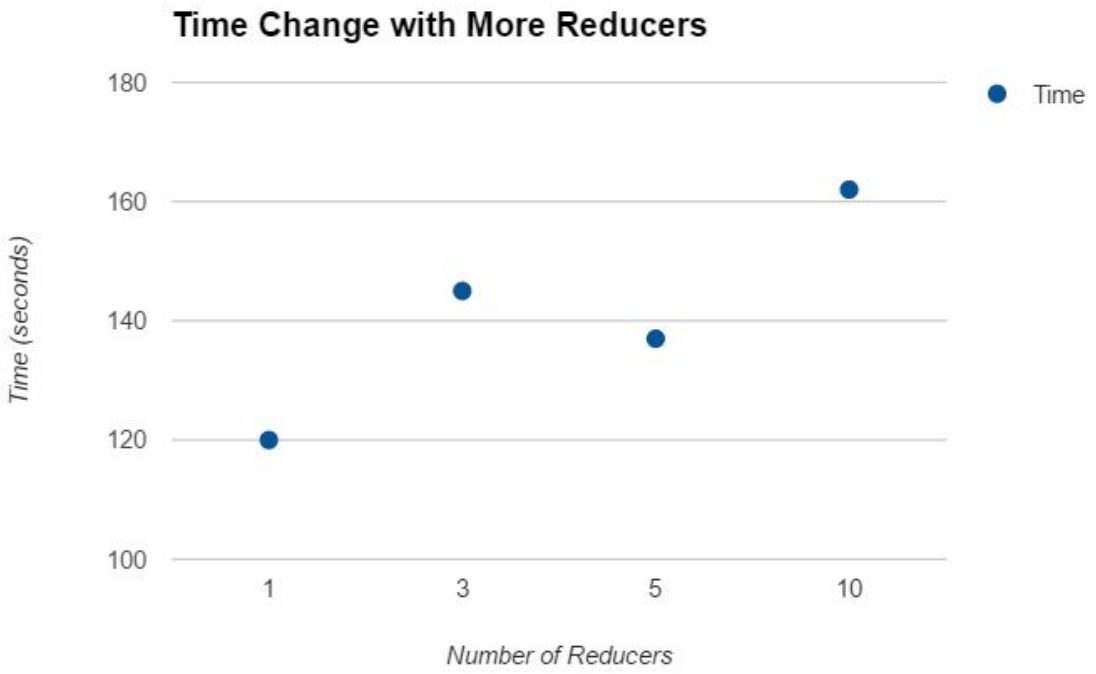
The MapReduce on the Hadoop cluster took 64 seconds to complete when we ran it on the 5.1 Gigabytes of data.

After the MapReduce, there were 13,182 lines in the file that was output. One line for each location. This is the number of rows it contributes to the Cassandra table as well. The file was 385,765 Bytes in size.

It took 2.799 seconds to create and populate the Cassandra table with the 13,182 rows of data. On average, the system loaded 4,709 rows/second.



Here is a graph that shows how much time it takes with different sized inputs. We did not change the number of reducers so that it would be a constant. It is visible that after a certain size, the amount of time does not grow as quickly. This shows how Hadoop is better suited for larger amounts of data.



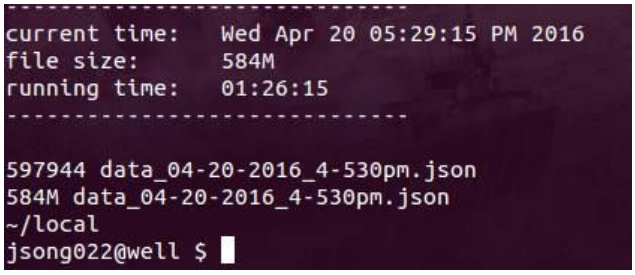
This graph shows the change in the amount of time it takes when you add reducers manually to the command. In this case we used all of the data each time to make sure that stayed constant. It seems that 1 reducer is the best option for the amount of data we were considering. It is also interesting to see that the time taken drops at 5 reducers but grows otherwise. This seems to be because 1 reducer is enough for our data and adding more results in more overhead.

In total we collected about 35 Gigabytes of data for this project. It took around 2.5 hours to collect each Gigabyte.

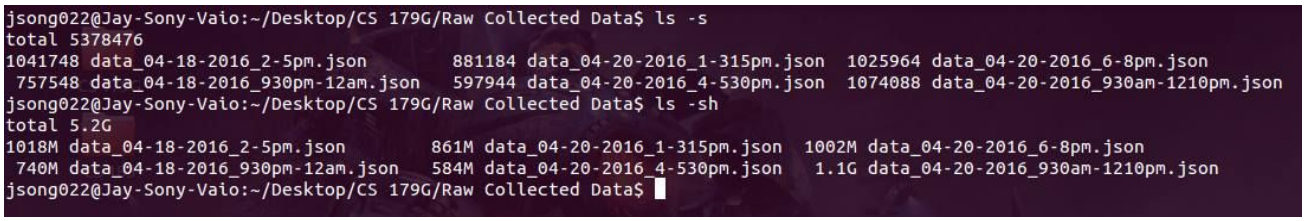
The table in Cassandra that contains the most popular hashtag by location has 119,694 entries. This does not mean we have that many locations because some of the locations are the same as others but have multiple entries for multiple days. This data was collected over 11 days.

The table in Cassandra that contains the latitude and longitude values of each hashtag that had coordinates, is populated with 2,854,691 entries. This data was collected over 11 days.

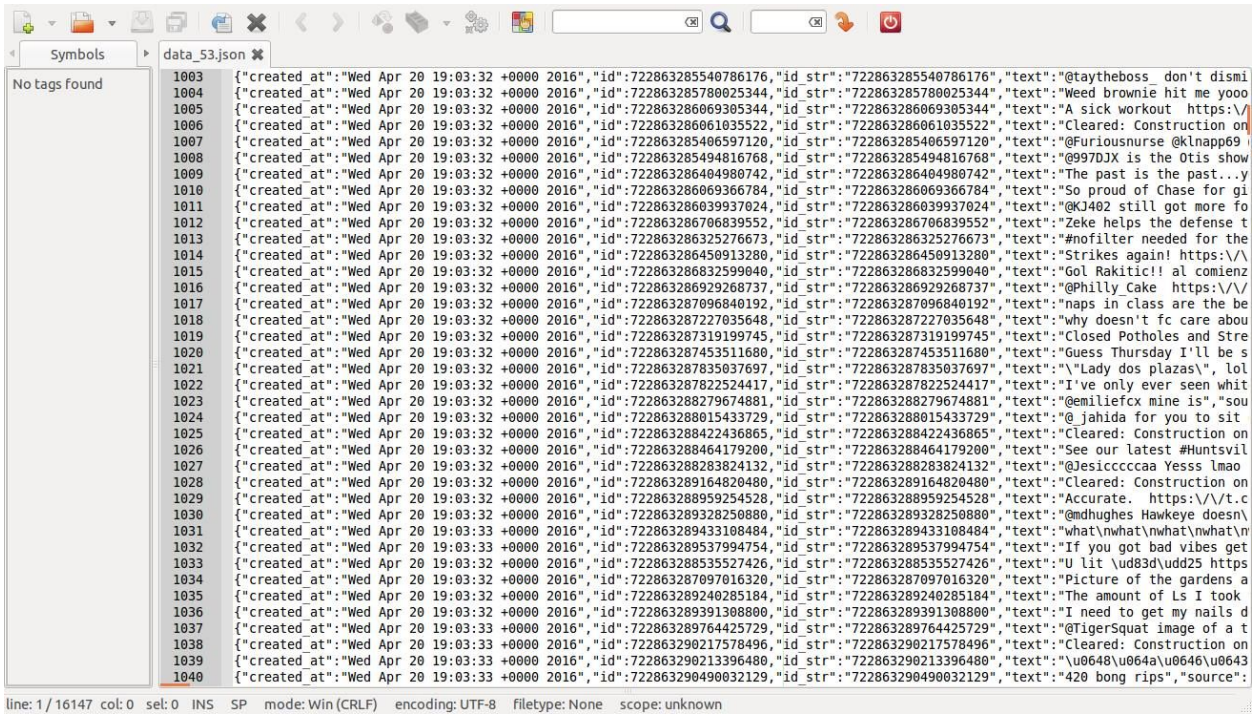
Screenshots



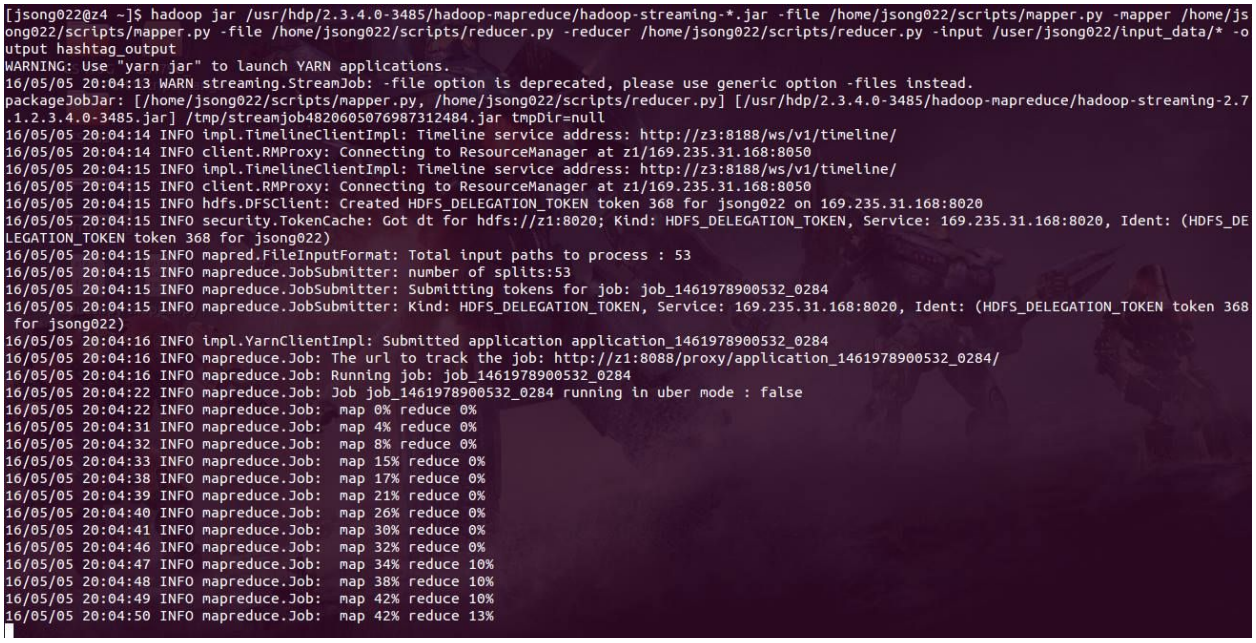
Here is one the data collection sessions that Jay Song conducted. It shows that he collected 584 Megabytes in about 1 hour and 26 minutes. It also shows the size of the file data_04-20-2016_4-530pm.json to be the same as the collected data. This shows that our script was correctly populating the file.



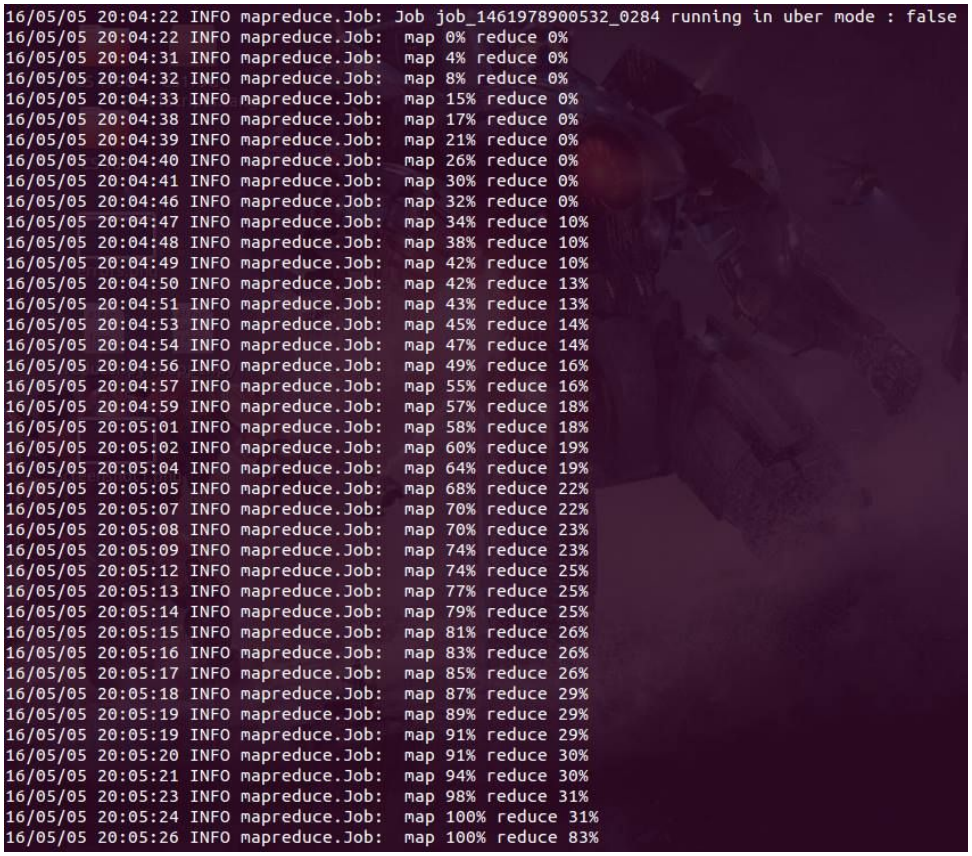
This images shows two of the files Jay Song used to collect data and the amount of data he collected in each.



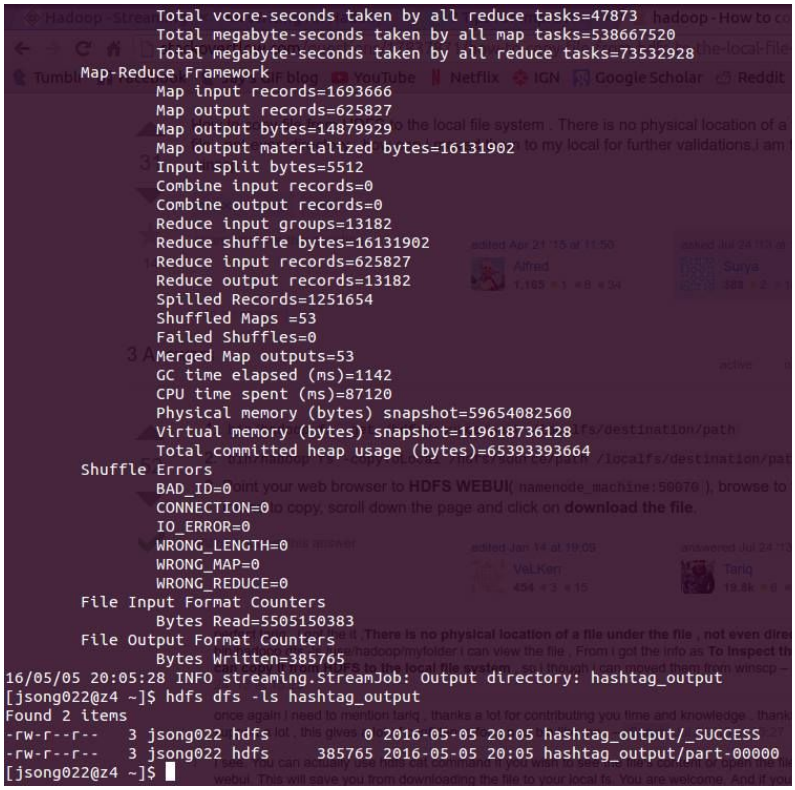
This shows one of the files created after removing the extra lines that did not contain a tweet.



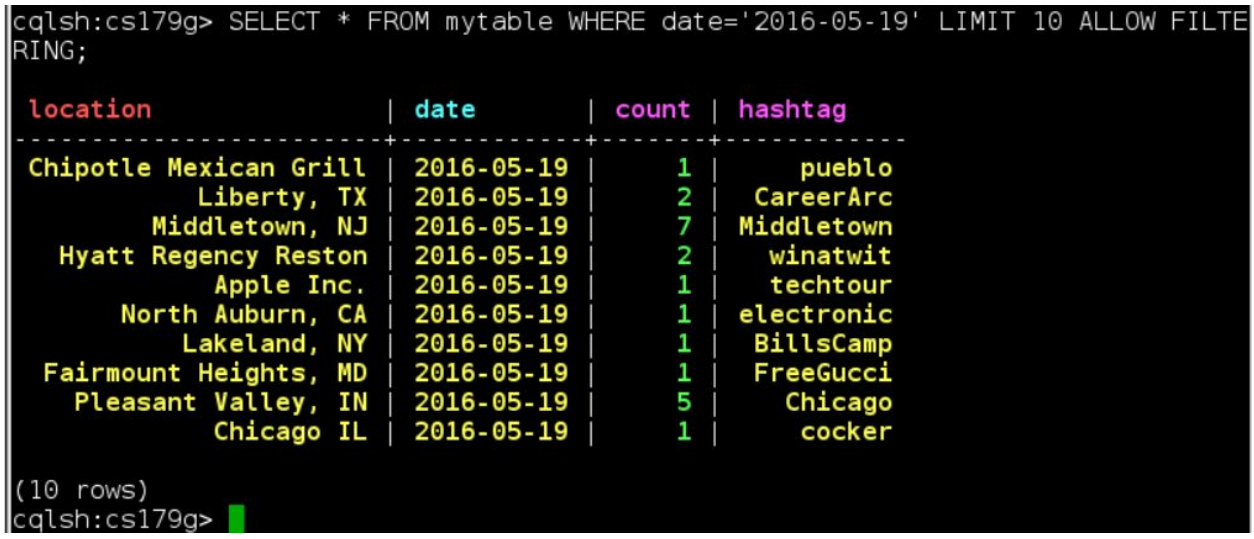
Here you can see the command use to run the Hadoop cluster using the mapper.py script and the reducer.py script. You can also see it starting to map and reduce. This command uses the hadoop-streaming jar so that the scripts can be written in a different language than Java. We used this to write the scripts in Python instead.



Here is a screenshot of it continuing to work. It finishes mapping and is almost done reducing.



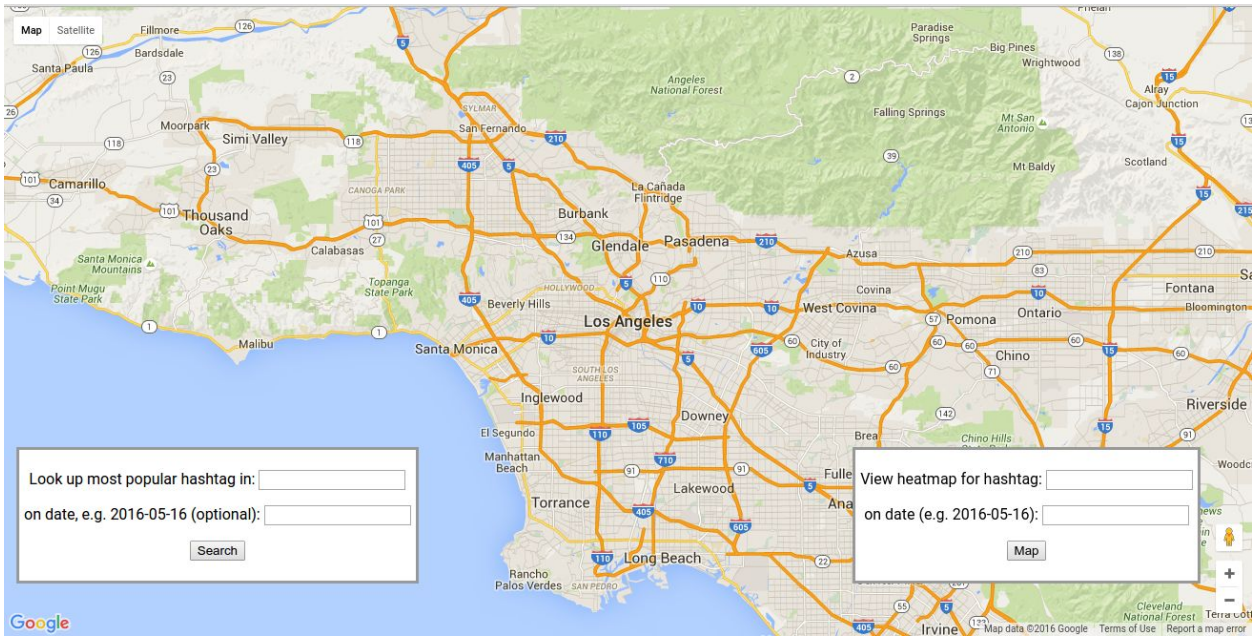
Here you can see the end of the MapReduce phase. We've also displayed the file it created called part-00000 that is 385765 Bytes. This file is what we change and use to populate Cassandra.



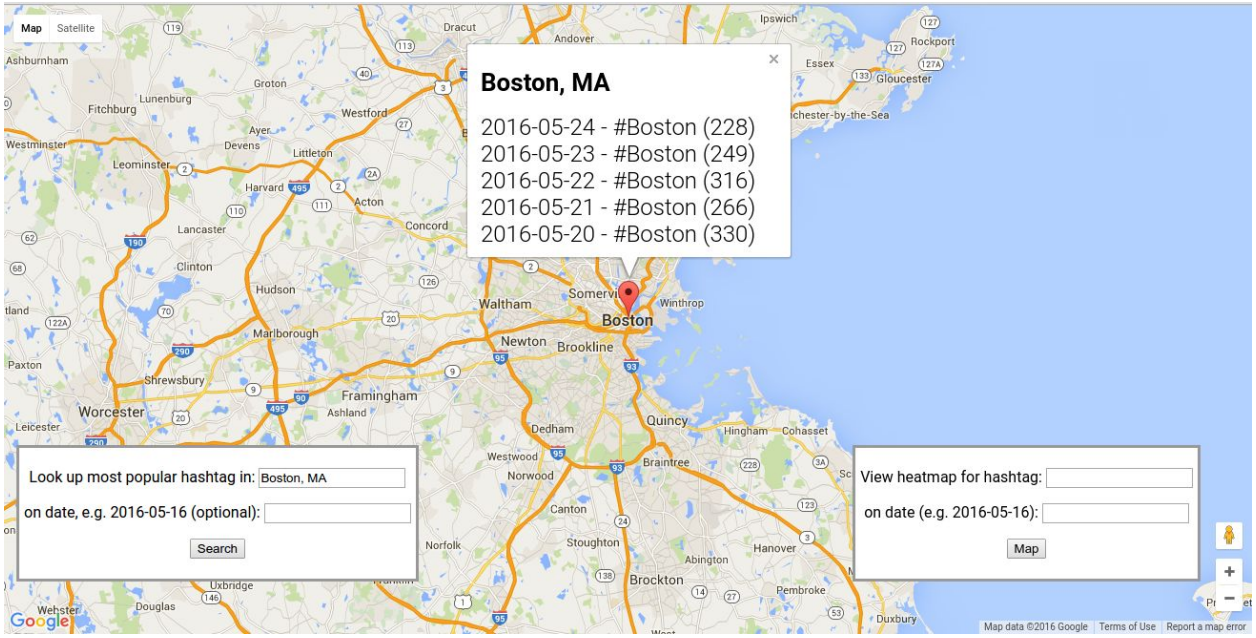
Here is a sample of what the mytable data looks like. This is the data that gives us the most popular hashtag in a city by date.

```
cqlsh:cs179g> SELECT * FROM htcrd WHERE date='2016-05-23' AND hashtag='Hiring' L
IMIT 10 ALLOW FILTERING;

 id    | date       | hashtag | location          | lat          | long
-------+------------+---------+-------------------+--------------+------------
 62693 | 2016-05-23 |  Hiring |         Boise, ID | -116.2146068 | 43.6187102
 51678 | 2016-05-23 |  Hiring |     Manhattan, NY |  -74.0130866 | 40.7045399
 54191 | 2016-05-23 |  Hiring |    Richardson, TX |  -96.7298519 | 32.9483335
 62602 | 2016-05-23 |  Hiring | Kingston, Ontario |  -76.4926865 | 44.2243515
 48451 | 2016-05-23 |  Hiring |    Manchester, NH |  -71.4547891 | 42.9956397
 63461 | 2016-05-23 |  Hiring |    Pittsburgh, PA |  -79.9958864 | 40.4406248
 64791 | 2016-05-23 |  Hiring |        Canton, MS |  -90.0662276 | 32.6068242
 65053 | 2016-05-23 |  Hiring |     San Diego, CA | -117.1654397 | 32.7692001
 64482 | 2016-05-23 |  Hiring |       Raleigh, NC |  -78.6381787 | 35.7795897
 47076 | 2016-05-23 |  Hiring |        Eugene, OR | -123.0867536 | 44.0520691

(10 rows)
cqlsh:cs179g>
```

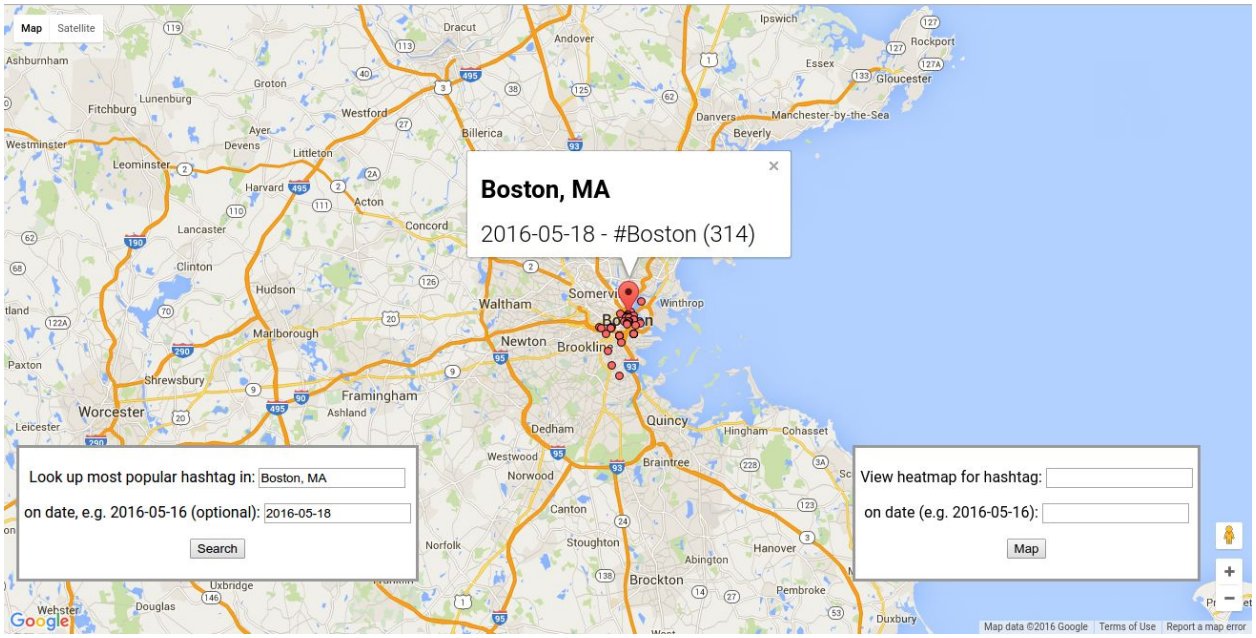
Here is a sample of what the data in the htcrd table looks like. This table gives us the coordinates for each hashtag and is used for the heatmap as well as the dots to mark where hashtags were sent from.



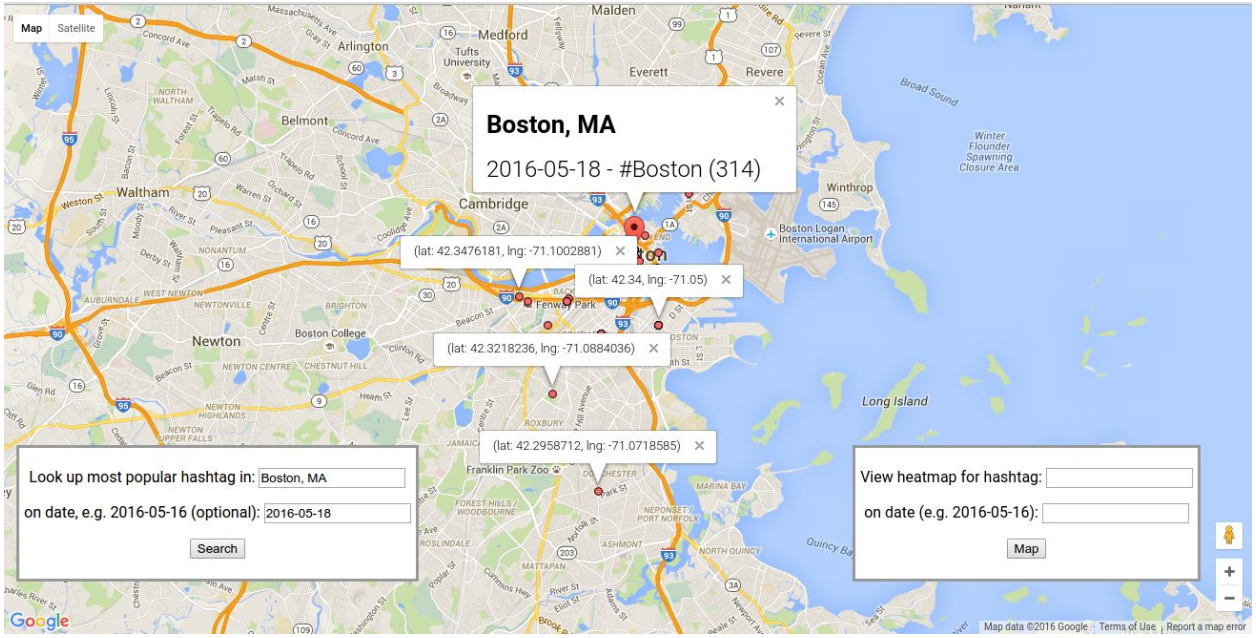
Here is the home view of the Ruby on Rails application. This is what you see when you first visit the page.



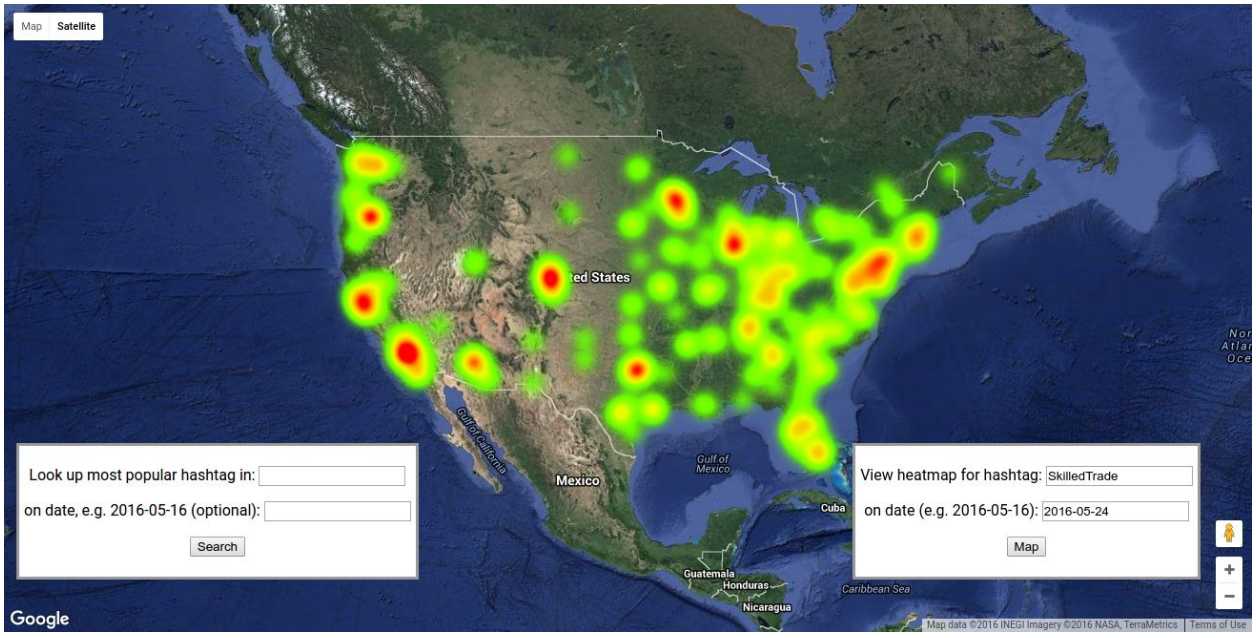
Here is the search view when the location specified is Boston, MA. You can see that it gives you 5 values of the most popular hashtag in the city. These values will be the 5 most recently input values in the database. This screenshot was taken on the 24th of May, 2016 and so it has these dates as the most recent.



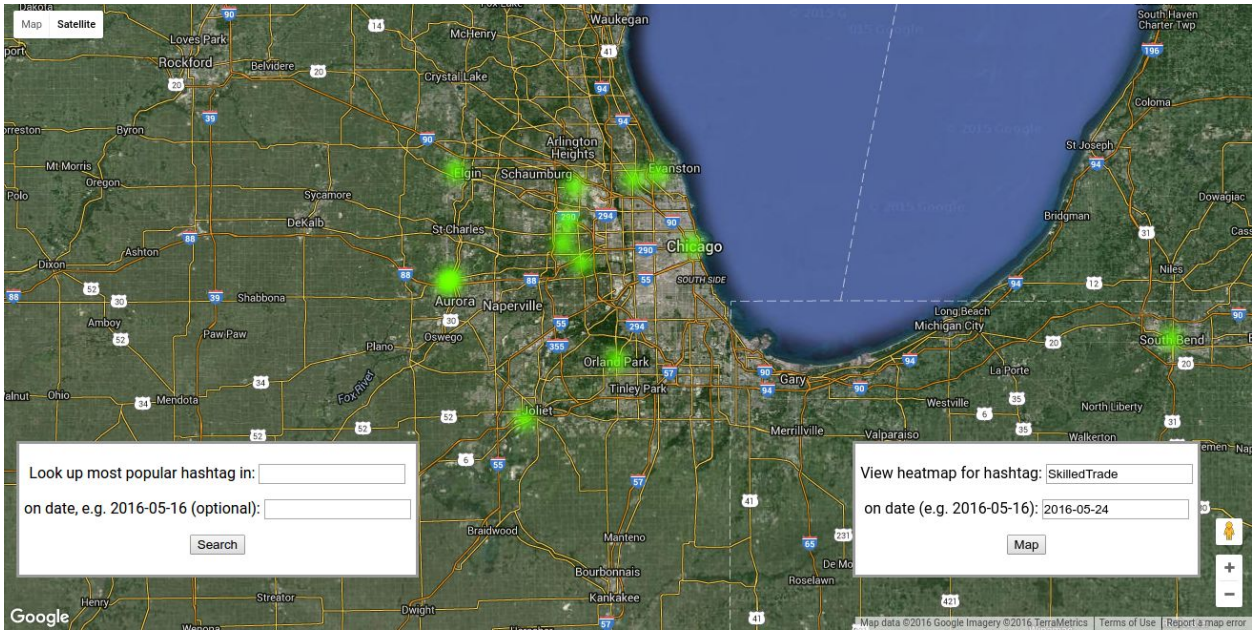
Here you can see the search view when a valid location and date are specified. In this case it is Boston, MA on May 16th, 2016. It lists the most popular hashtag on that day. It also marks where the tweets that contained that hashtag were posted from in Boston, MA with little red dots.



One of the features of the search view is that you can click on the little dots on the map and it will give you the exact latitude and longitude values of that point. You can see here that the user has the ability to click on any number of dots. You can also click on an unmarked point on the map to close all the opened information boxes.

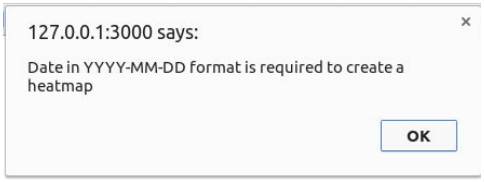


Another feature of the map is the ability to see the distribution of a hashtag on a given day. In this case we use the hashtag SkilledTrade on May 24th, 2016. You can see where there is a higher concentration of people that used this hashtag because the map is red in those places.

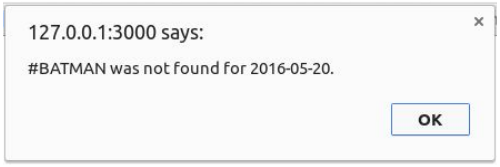


Here is view of how the heatmap behaves when you zoom into an area. This is for the hashtag SkilledTrade on May 24th, 2016. This allows you to get a more specific idea of how the heatmap is actually distributed over smaller areas.

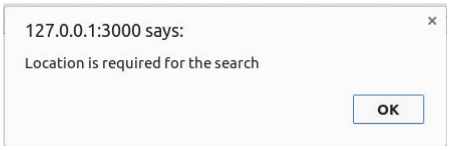
This is the error message that will pop up if a hashtag is not specified or is invalid for the heatmap.



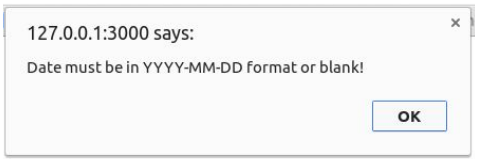
This is the error message that will pop up if the date given for the heatmap is invalid.



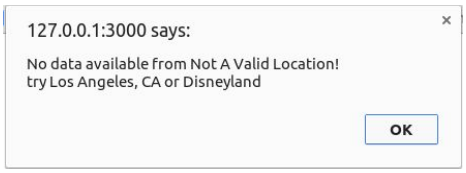
This is the error that will pop up if there is no data for the specified hashtag on that specific date.



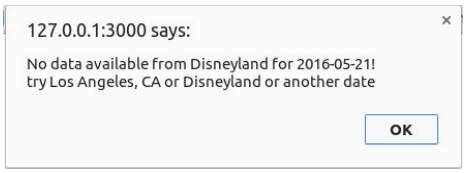
This is the error that will pop up if no location is specified when trying to find the most popular hashtag.



This is the error that will pop up if the date input is not in the correct format.



This is the error message that will pop up if the location is not found in our database.



This the error that will pop up if there is no data for the specified date.

Contributions

Each of us did separate research on Java, Python, and the libraries needed to connect them to the Twitter Streaming API for Phase 1. After that we came together and used what we had learned to figure out how to best obtain the data.

Jay Song created the account we needed in order to use the Twitter Streaming API.

Next, Palak Parwal, Jay Song, and Michelle Wong came together and wrote a script in python that would collect the stream of incoming tweets and save them to a file. All three of them worked together to debug and test the code, as well as make sure the results were correct.

Palak Parwal and Jay Song took this python script and ran it separately to more quickly obtain the 5 Gigabytes of raw data needed. Together, they obtained a total of 5.2 Gigabytes.

After looking through the data, Palak Parwal, Jay Song, and Michelle Wang, wrote a script that would cleanup the data and ignore unnecessary lines from the files while breaking it into smaller chunks.

Jay Song took this script and created the 53 files of raw data that now totaled 5.1 Gigabytes.

Michelle Wang uploaded the 5.1 Gigabytes of data to the lab servers so we could use it in the coming parts of the project.

Palak Parwal, Michelle Wang, and Jay Song came together to write the mapper.py script and reducer.py script for phase 2 of the project.

Jay Song took these scripts and ran them with the Hadoop cluster to obtain the data file with the location, most popular hashtag, and number of times the most popular hashtag appears.

Palak Parwal configured Cassandra to make sure it would function correctly for the remainder of the project.

Michelle Wang and Palak Parwal wrote a script that would create a table in Cassandra and populate it. They ran it to populate the table.

To add more data to this project, Palak Parwal, Jay Song, and Michelle Wong created a set of scripts that would collect more data, parse it, MapReduce it, append data to it, and upload the data to Cassandra tables. These scripts include twitter_streaming.py, mapper.py, and reducer.py. They are joined by endtwitter.sh, importdailydata.sh, and mapper2.py which were written by all three members together. Jay Song ran these scripts daily.

Palak Parwal, Michelle Wang, and Jay Song created the Ruby on Rails application together to display the data from this project.

Palak Parwal wrote the report for this project.