High Performance Tweet Analysis using MPI

Peter Annable, Engineering 517 High Performance Computing, Spring Term 2018

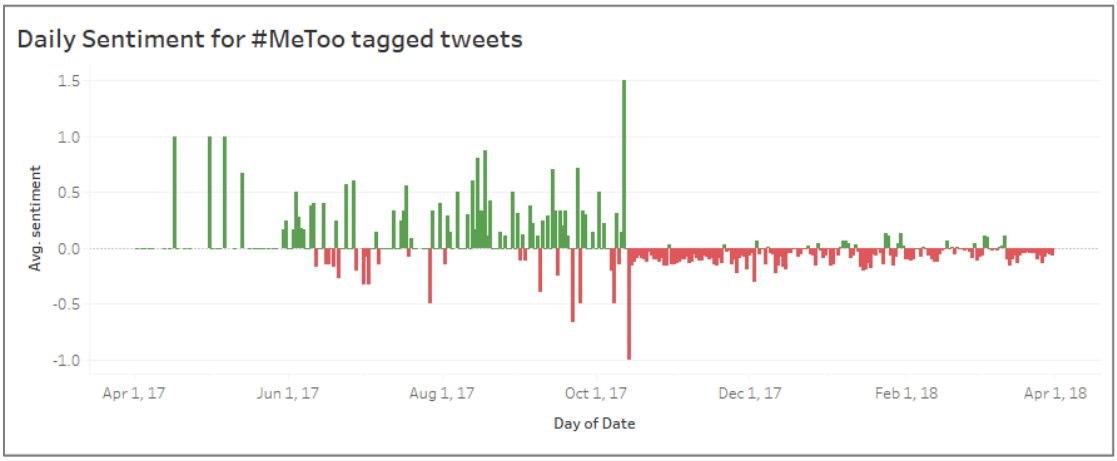

Fig. 1. – Daily Average Sentiment of Tweets tagged with the #MeToo Hashtag

Abstract – Analysis of twitter activity is a good proxy for understanding overall sentiment on topical issues, brands or marketing campaigns. Often such analysis can involve processing of millions of tweets to understand sentiment over longer periods of time. This paper demonstrates how a Message Passing Interface (MPI) program can be used to allow analysis of a large corpus of tweets over large numbers of processors. Using twitter data from the Indiana University Observatory on Social Media project to gather large numbers of tweets based on selected hashtags, strong scaling of tweet analysis is shown using up to 256 processing cores.

Index Terms —Twitter, Sentiment Analysis, MPI, Big Red II

Introduction

This project is inspired by the Pulse of the Nation, an analysis of tweets in the United States that was used to infer the mood of the people and use an animated map to show how the mood shifts over a period of 24 hours. In a similar manner, marketers are interested to understand how a hashtag and the sentiment about that hashtag moves over a period of days or months. To understand this, a large collection of tweets that contain a specified hashtag over a period of several days or weeks will be extracted, analyzed for sentiment and visualized.

The following insights are expected from this project:

1. Scalability of sentiment analysis approach using MPI Programming.
2. An understanding of how sentiment changes over time for long running topics or hashtags on twitter.
3. Potential conclusions that can be reached from the analysis of use to researchers or marketers.

1 Method

1.1 Solution Architecture

The Total solution developed consists of 5 main components, as shown in Figure 2:

1. Data acquisition from the OSoMe tool "Moe's Tavern"
2. Transformation of JSON lines twitter data to csv file.
3. Sentiment analysis on Big Red 2.
4. Creation of data output at end of sentiment analysis
5. Visualization of the data using Tableau.

1.2 Data Acquisition from Moe's Tavern

Moe's Tavern is a website that provides access to public tweets collection, representing 10% of all pubic tweets on the twitter.com website. Moe's Tavern is one of the tools offered from the Indiana University Observatory on Social Media (OSoMe). Approval from the OSoMe team must be requested to access this site.

To acquire large collections of tweets, I performed several searches on hashtags and phrases. Below are a few of the key data sets used:

Search Term	Time Period	Tweets Returned
#GoBlue	1/1/18 – 4/10/18	60k
#MeToo	4/20/17 – 4/22/18	774k
#NationalSchoolWalkout	1/1/18 – 4/22/18	42k
Colin Kapernick (text match)	9/1/17 – 4/20/18	376k
#LoveOverBias	1/1/18 – 4/20/18	12k
#BlackLivesMatter	9/1/16 – 4/20/18	664k

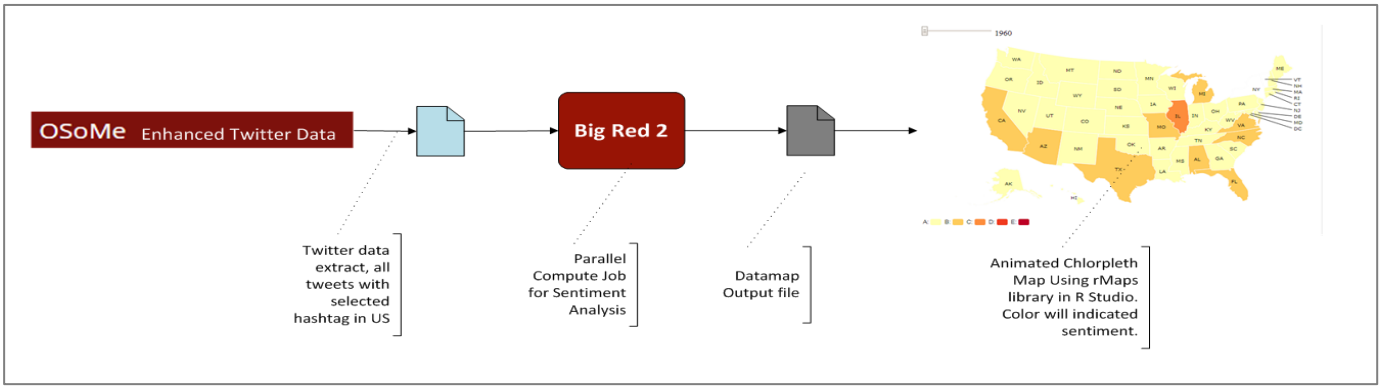

Fig. 2. – Solution Architecture Overview of Tweet Analysis Process

Moe's Tavern delivers the results for download via gunzip files, usually within a few minutes, but larger queries can take an hour or more.

1.3 Data Preparation

Tweet data from Moe's Tavern is returned in a JSON lines format. This is fairly complex structure with nested levels in the JSON to represent the entire tweet structure. This structure is documented on the Introduction to Tweet JSON page on the twitter developer's site.

To avoid the need for developing a complex C routine for parsing this format, a UNIX script was developed based on the command-line json processor jq. This script translates each JSON line into a CSV formatted file with the following columns:

1. Tweet Id
2. Tweet Date
3. Location of User
4. Number of Retweets
5. Hashtags in tweet
6. Tweet Text

The basic syntax of the processing script is:

```
jq -r -f ./cmds.jq $1
```

Where $1 is the file to scan, and cmds.jq is the json query language file developed to perform this process. The syntax is somewhat similar to awk. Given that .csv files are not very robust in terms of handling special characters and newlines, the processing including ensuring that all quotations and newlines were removed from each tweet field:

```
{id: .id_str, date: .created_at, location: .user.location,
retweets: .retweet_count, hashtags: .entities.hashtags,
text: .text} |
{id: .id, date: .date, location: .location, retweets:
.retweets, hashtags: [.hashtags[].text], text: .text} |
{id: .id, date: .date, location: .location | tostring |
gsub("[,\n\"]"; " "), retweets: .retweets, hashtags:
.hashtags | join(";"), text: .text | gsub("[,'\"\n]"; " ")}
|
[.id, .date, .location, .retweets, .hashtags, .text] |
@csv | gsub("[\"]"; "")
```

After the .csv file is saved, the final step is to remove all other non-ascii characters to prevent potential problems in the C code string process that will be used. This is done with the UNIX tr command:

```
tr -cd '\11\12\15\40-\176'
```

The final script, "jparse.sh" that performs these processing steps is included in the assignment submission. For large extracts, Moe's Tavern usually provided multiple large JSON files. A small script was used to process each in turn and merge into a single .csv file. All file processing steps were done an Ubuntu 14.04 LTS desktop, and resulting csv files transferred to Big Red II.

1.4 MPI Analysis Program

The heart of the analysis process is analyze_tweets_mpi.c. This program was developed in a single threaded form first, then translated to the MPI version.

1.5 Serial Process Development

The overall process to score tweets is straight forward: read the .csv file of summary tweet information in, score the text of each tweet for positive and negative sentiment, and write an output .csv files with the tweet information and scores that can be later consumed by visualization software.

In fact, this part of the process took the most time to develop as managing memory properly in C was critical to the process large numbers of tweets in-memory. Several days were spent identifying the source of Segmentation Fault errors and ensuring the code was free of any memory or pointer usage problems before converting to MPI.

The first step is to define a struct for a tweet:

```
#define TWEETSIZE 280
struct tweet {
    char   id[40];
    char   date[80];
    char   location[80];
    int    retweets;
    char   hashtags[80];
    char   text[TWEETSIZE];
    char   keywords[TWEETSIZE];
    double pos_score;
    double neg_score;
};
```

Knowing that I will later be passing tweets using MPI routines, I used a fixed structure size to simplify data passing.
TWEETSIZE was set at 280 characters to accommodate the recent increase in maximum tweet size.

The next step is to determine the number of tweets in the file so we know the amount of memory to allocate. This was done using a simple routine that opens the file and reads each line until end, returning the count:

```
Int count_file_lines(char* filename)
```

This gives us the total ROWS to include in our memory allocation. Calloc() was used to ensure the memory is initialized to zeros.

```
matrix = (struct tweet*) calloc(ROWS,
sizeof(struct tweet));
```

Once I have an array of structures allocated, and I know how many lines to read, I use a routine read_tweets() to read and parse the csv file. Here, early attempts at coding this manually were inadequate and I suffered from segmentation fault errors if there was no data at a given csv column. In test data this wasn't seen, but with real twitter data, many locations were blank and couldn't be handled by my code. To improve upon this, I imported a more robust csv reader, tinycsv, from https://ideone.com/mSCgPM. This routine was developed by Deligiannidis Konstantinos and is distributed under the GNU General Public License. This fixed my problem with null columns. Lastly, each tweet text is converted to lower case to simplify later matching with keywords.

The next step is to load in two sets of keywords for positive and negative sentiment. These were obtained from the Opinion Lexicon at Illinois University at Chicago.
https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
This resource has over 6800 English words classified as positive and negative for sentiment analysis.

Two files were created: positive-words.txt and negative-words.txt. These were loaded in a similar manner as the tweets, into an array of "phrase" structs:

```
struct phrase {
    char text[PHRASESIZE];
    double score;
};
```

A default score of 1.0 is used for each word.

With the tweets and comparison phrases loaded, we can now process each tweet to score it. A simple algorithm was used, which is to loop through each word in the tweet text, and look for matches in the positive and negative phrase list. If a match is found, the respective positive or negative sentiment score is incremented, the word found is added to a new list of keywords, and the code breaks from the loop. The essential code is here from the score_tweet() function:

```
while (token != NULL) {
   if (strlen(token) > 1){// skip single char words
      for (i=0; i<plines; i++) {
         if (strcmp(token, p[i].text) == 0) {
            // found a match
            if (mode == NEGATIVE)
              score-=p[i].score;
            else
              score+=p[i].score;
            strcat(t->keywords,p[i].text);
            strcat(t->keywords,";");
            break;  // found match so break
           }
        }  // end for
     } // end if
    token = strtok_r(NULL, sep, &saveptr);
}
```

After all tweets are processed, the tweet structs are now updated with a pos_score, neg_score, and list of keywords. The final step is to write a new .csv file with the tweet Id, Date, Location, Scores and keywords. This will be later used in visualization software.

1.6 MPI Version and Performance Considerations

I considered two possible strategies for the MPI version of this code. The first was to use the MPI File IO library to allow each process to open and read a section of the tweets data file. Then upon completion of processing, write back to a shared output file. This would be advantageous for very large file sizes since the amount of data is evenly split between processes. The difficulty with this approach is that each line of the tweet file is variable in size. Since the file IO commands rely on a consistent offset for each process, some kind of additional processing would be required to identify which lines had been obtained and process them without duplication. Given the largest file size I expected to read would be 1 GB, this seemed unnecessarily complex.

The second strategy considered and implemented for this assignment is to read the complete file in on the ROOT rank, and load each line into the defined tweet struct. Since I know each tweet will now consume the same amount of memory, MPI_Scatter() can be used to divide up the work based on the number of tweets. Once the processing is complete and the scores added to the tweet structs, MPI_Gather can be used to bring all the results back to the ROOT rank for writing out a single output file.

The chosen approach was quite simple to implement, once a custom MPI Datatype is defined. The disadvantage is potentially being inefficient in broadcasting a large amount of data back to and from each process. However, given this is only done once at the beginning and once at the end, and no communication during the processing, this process seemed like a good approach and would likely accommodate processing of millions of tweets at a time.

The final main routine in pseudo code is:

```
read_phrases()
MPI_Type_create_struct() and MPI_Type_commit()
if (rank == ROOT) {
   read_tweets();
   calculate number of tweets per proc;
```

```
   calculate remainder of tweets after distribution;
}
MPI_Scatter();  /* from root to workers */
score_tweets();
MPI_Gather();   /* from workers to root */

If (rank == ROOT) {
   Write_output();
}

MPI_Finalize();
```

1.7 Output Format

The final output file is written to a csv file with the following columns:

- Tweet Id
- Date and Time
- Positive Sentiment Score (>=0)
- Negative Sentiment Score (<= 0)
- Location of Tweet (if present)
- Sentiment Keywords found, separated by semicolons

This format is ready for consumption by external visualization programs such as Tableau.

2 Results

2.1 Data Sets Analyzed

For scalability testing I used the largest data sets I could obtain from Moe's Tavern. This was a search for the #MeToo hashtag for the last 12 months, which produced 776,198 tweets. The other data sets mentioned in section 1.2 were also processed and visualized. Example visualizations and discussion of the results follow later in this paper.

2.2 Weak Scaling

The MPI program demonstrated fairly strong weak scaling performance as shown in Figure XX below. Of course, weak scaling will ultimately be limited by memory size on the root node, however I did not come close to approaching that.

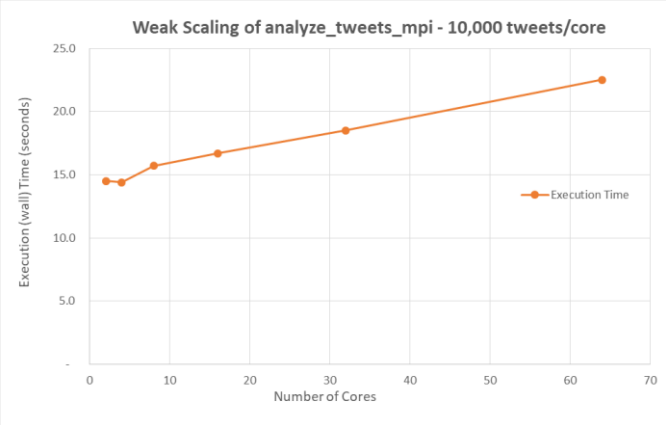

Fig. 3. – Weak Scaling from 2 to 64 cores

2.3 Strong Scaling

Testing with up to 256 cores revealed that total time to solution continues to drop dramatically until about 32, cores, and after 64 virtually flattens out. The best total execution time was 19.2 seconds at 256 cores with a problem size of 776,198 tweets, with an input file of 139MB.

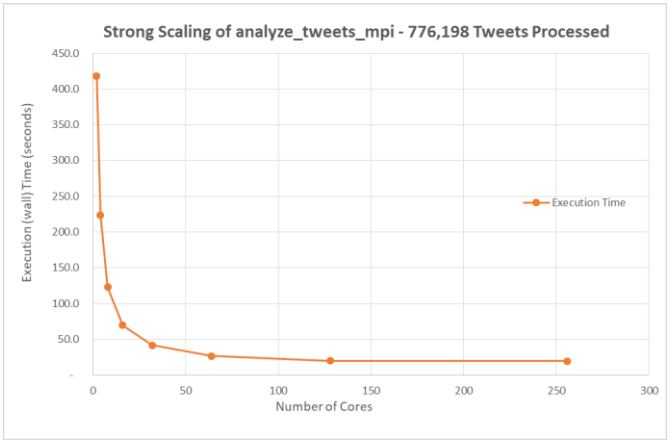

Fig. 4. – Strong Scaling Performance from 2 to 256 Cores

2.4 Memory Access Bandwidth

The analyze_tweets_mpi program is memory intensive. All tweets are read into memory, and each tweet word larger than 1 char is compared with thousands of positive or negative keywords. All tweets are transmitted and gathered during the process as well.

To understand memory bandwidth consumed, the overall memory reads were compared with the HPCG benchmark. This benchmark, running on 16 cores, reported throughput of 1.53 GB/sec per core. 16 cores was chosen for the benchmark as it represented the best balance between time to solution and increasing overheads in the analyze_tweets_mpi program.

To estimate memory bandwidth of analyze_tweets_mpi, the total bytes read in during processing was estimated with the following data, using the 16 cores benchmark run on the MeToo data set:

25,548	average keyword chars per tweet
45,281,128	Total tweet chars
1,156,842,258,144	total chars read from memory
139,245,648	total bytes read from file at initailization
1,335,060,560	bytes broadcast and brought back
1,158,316,564,352	**TOTAL Bytes Read**
1,041,651,587	Bytes/Sec/Processor (16 processors in 69.5 secs)

Total memory throughput was 0.97 GB/sec vs. 1.53 GB/sec for the HPCG benchmark. Given that the 0.97GB/sec is an average number through the life of the program execution, it's likely the peak was higher and we can assume memory bandwidth was the main constraint on this applications performance.

2.4 Visualizations

The primary visualization produced is shown in Figure 5, which shows average sentiment per day, color coded with green as positive and red as negative. Below this graph is the volume of tweets per day. What can be readily seen here is that the #MeToo hashtag had very little volume until Oct 16, 2018 when there was a spike in volume. This was the date that the news about Harvey Weinstein was published, creating the movement. As can be seen, overall sentiment was negative after that time.

Figure 6 shows a similar analysis for the #BlackLivesMatter movement. This demonstrates several peaks in volume, likely associated with external events. Sentiment is much more strongly negative with this hashtag.

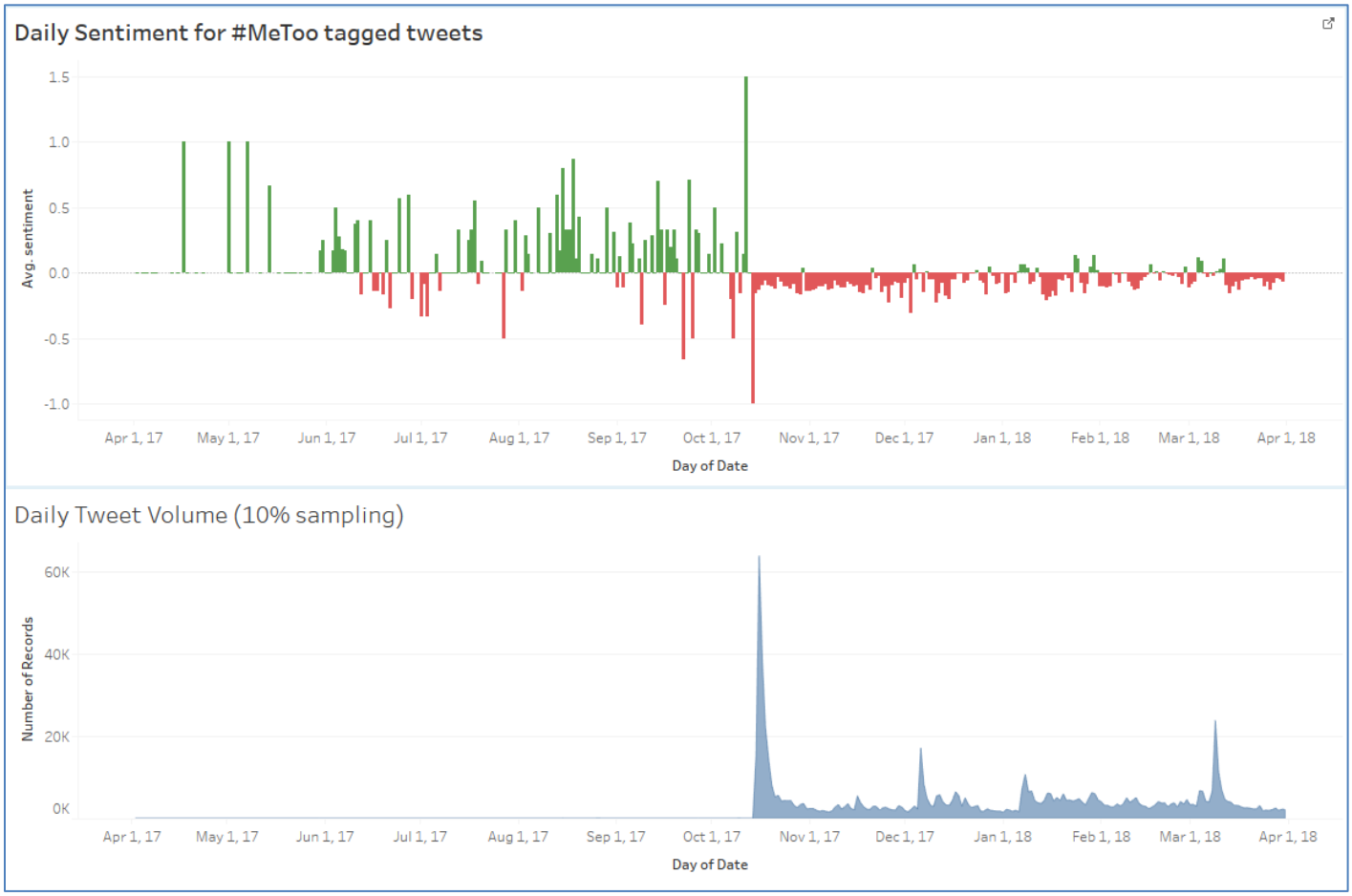

Fig. 5. – Daily Sentiment and tweet volume for the #MeToo hashtag.

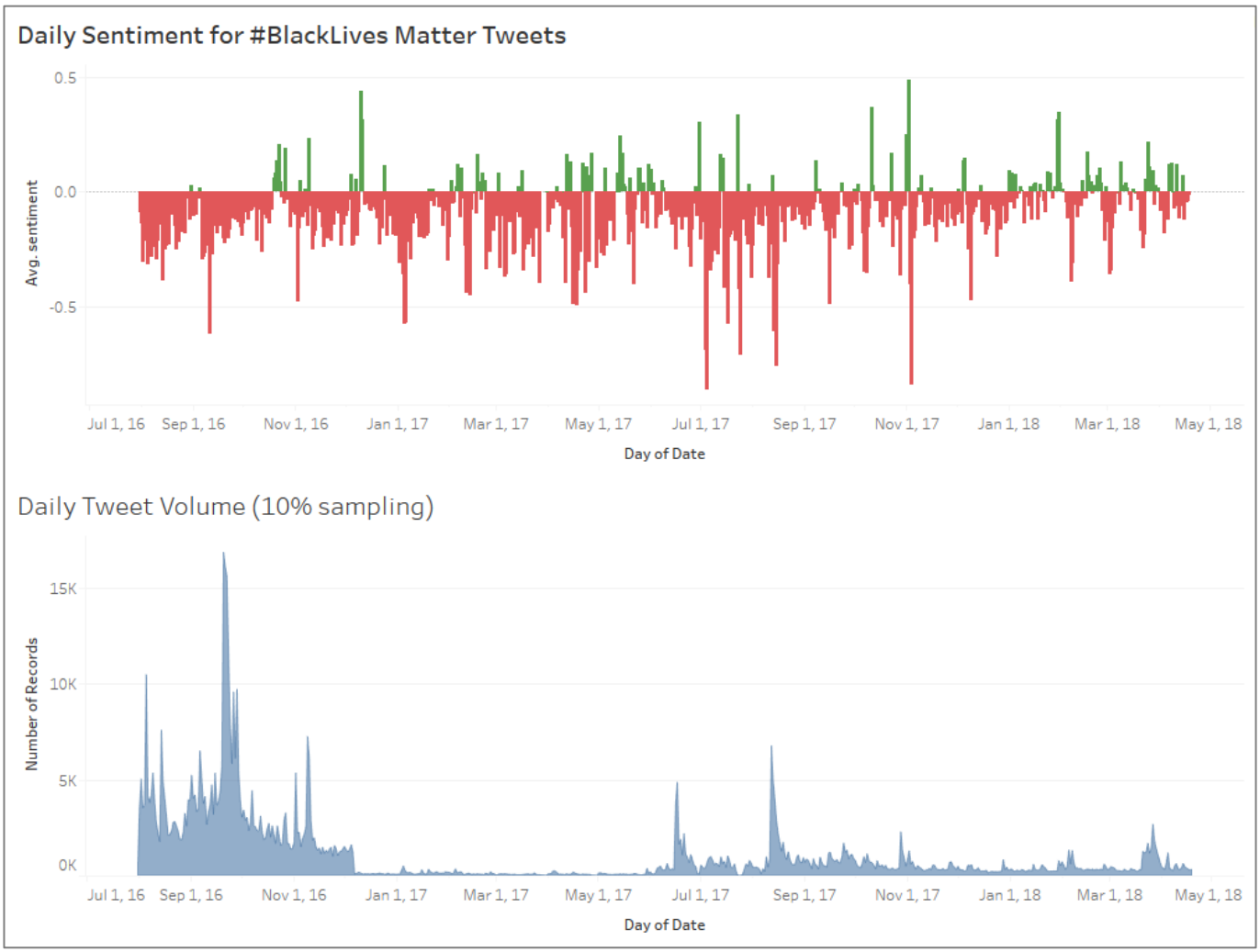

Fig. 6. – Daily Sentiment and tweet volume for the #BlackLivesMatter hashtag.

3 Discussion

3.1 Potential Improvements

There are a few potential improvements that could be made to this process to improves its efficiency and effectiveness. The first would be to replace the brute-force string matching of keywords with a simple tree search to more quickly identify keyword matches. The second would be to replace keyword matching with a more robust machine learning algorithm trained on tweet sentiments. Given the importance of context and vocabulary, this would likely produce more reliable results.

To improve scalability using the MPI File IO routines to divide tweet processing would allow for much larger file sizes. One potential approach would be to create fixed string lengths in the files as the csv is generated. This would allow the file IO routines to be used more easily. Upon reading in, extra spaces would not matter in the sentiment analysis process and could be trimmed off.

3.2 Geo-Temporal Analysis

An original goal of this report was to show a geographic representation of twitter sentiment and animate it over time to show change in a visual way, similar to the "Mood of the Nation" report. However, as was learned during the process, very few tweets now have geo-encoding to make this easily possible. This appears to be due to greater privacy concerns from when the Mood of the Nation report was done. A user-reported location field is available, but as a free text field, it was found to have very messy data and many unusable locations reported. Overall, this project still produced useful analysis in understanding sentiment over long periods of time.

Acknowledgements

The author wishes to thank Dr. Matthew Anderson and Tim Gilmanov for their help in this project design and throughout the E517 High Performance Computing course.

References

[1] Indiana University On Social Media. (OSoMe) Moe's Tavern Twitter data source: http://osome.iuni.iu.edu/about/

[2] Tiny CSV Reader, © 2015 Deligiannidis Konstantinos, available at: https://ideone.com/mSCgPM.

[3] The Opinion Lexicon. (Hu and Liu, KDD-2004). https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon

[4] Mining Twitter for Consumer Attitudes Towards Airlines. Jeffrey Breen, July 4, 2001. https://datamatters.blog/2011/07/04/twitter-text-mining-r-slides/

[5] Tableau Data Visualization Software (Desktop version 10.2) https://www.tableau.com/products/desktop