Coronavirus Sub-hotspots Trend and News Fusion in United States
ELEN 6889, Large-scale Steaming Processing

Changxu Luo
Columbia EE
cl3875@columbia.edu

Yanlin Liu
Columbia EE
yl4238@columbia.edu

Yanwen Jing
Columbia EE
yj2556@columbia.edu

Weihan Chen
Columbia EE
wc2681@columbia.edu

***Abstract* — The coronavirus is breaking out globally. Information about COVID-19 is overwhelming but generally limited. In this project, we hope to build a more comprehensive information platform for users by doing stream processing to get sub-hotspots trends and fuse news. We used Google Cloud Platform and Twitter API, utilized Spark and Django, applied batching, reordering and some other optimization methods to our streaming algorithm, and finally successfully produced a real-time Web application displaying the epidemic situation in the United States and recommending relevant Twitter to users[1].**

***Keywords* —** Stream Processing, Spark, Cloud, Real-time Data Mining

1. Problem Statement

The COVID-19 pandemic has been threatening the whole world since its outbreak a couple of months ago. Currently, the United States has the largest number of confirmed cases, attracting the global attention. Many individuals and organizations are contributing to helping people through this difficult time together. Some organizations like JHU provide its Coronavirus map[5], containing information of the confirmed cases, deaths and their trends in different places. They also share the data repository to the public[4]. Some other organizations public news or government announcements through twitter and many other social medias.

But the current information sources have some limitations. They do great jobs in specific areas, but may be helpless in other fields. For example, a coronavirus map only has numbers. Users know how does the confirmed cases increase compared with yesterday, but all he has is only the number.

To provide everyone with a highly integrated information platform, we use the knowledge of streaming processing to analyze the sub-hotpots trends and corresponding news in United States from the Twitter. We filter the Twitter stream[6] and use spark to find the hottest hashtags in the filtered stream on Google Cloud Platform. We also query data of COVID-19 confirmed cases and deaths, and all these information are displayed in our front-end website, which is an US map with detail information of each state as well as news and tweets related to state's most famous topics.

In the following parts, we will firstly introduce the implementation of our project. Next, we talk about the streaming algorithms and optimization we use. We will then show our results by recapping the demo we did, and finally we will discuss about our project and list future work directions.

2. Implementation Description

In this part, we will introduce more details about our implementation. Figure 1 shows our system architecture and how we implement our project.

First, we use Twitter API to get streaming data and use filter to only retrieve tweets written in English, tweets containing the keywords we want, for example, Covid 19, Coronavirus. We also use the US states location data to filter the streaming data, make sure that we can retrieve tweets

[1]Source code can be found at https://github.com/camelboat/6889_ELEN_Large-scale_Stream_Processing_Project.

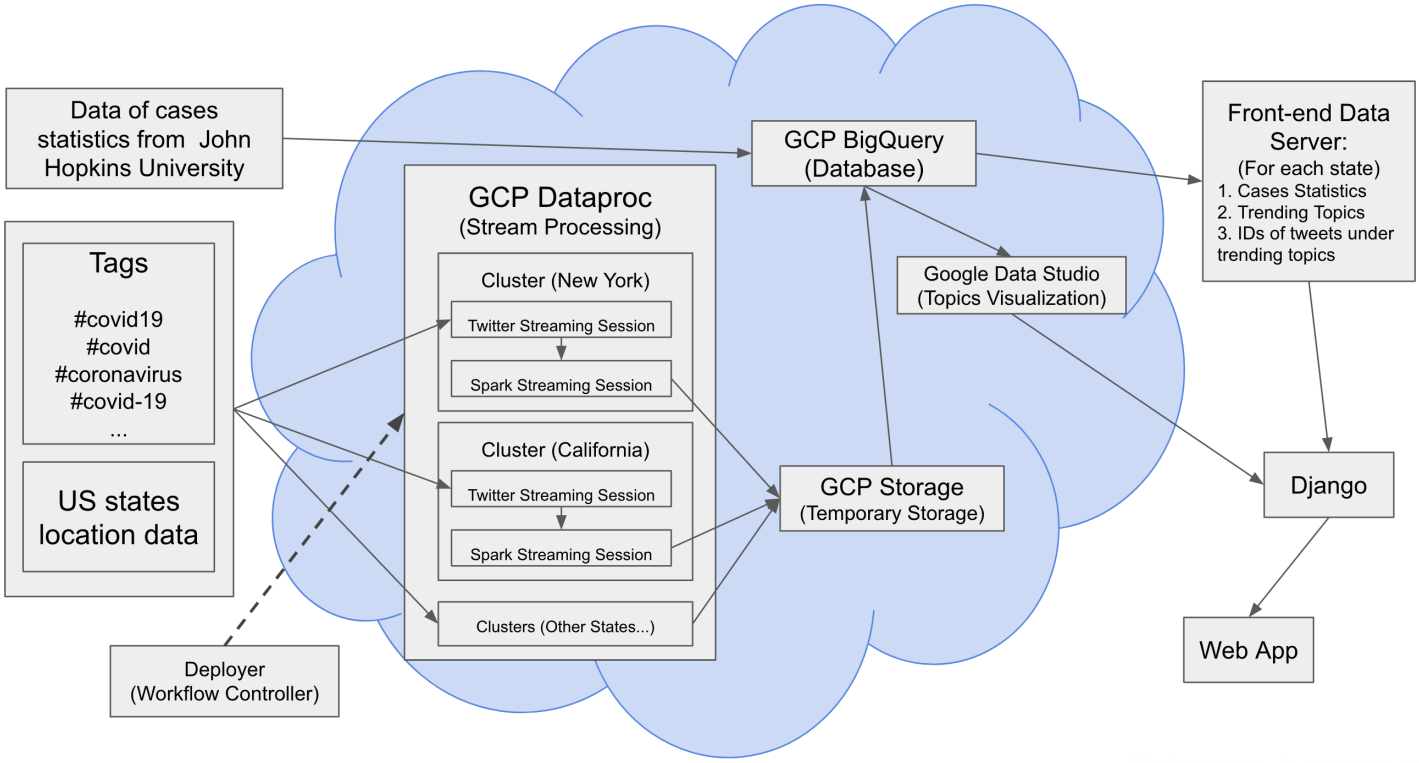

Figure 1. System Architecture

from a specific state. The location data is got from a University of Illinois public source[7]. We deploy our task on GCP (Google Cloud Platform) by using Workflow Controller, get the tweets from each state and use spark to process these steaming data. We count the tags that appear in the last three minutes with the highest frequency, then store these tags temporarily in GCP storage and in the end store them in GCP Bigquery.

John Hopkins University upload their dataset about the coronavirus to the GCP Public Dataset, we retrieve the information in this dataset by using Bigquery.We get the confirmed cases and death cases from this public dataset. This dataset is updated once a day, so we also update the cases information every day. For each state, we have the data about the cases statistics and the trending topics in Bigquery. In the next step, we retrieve some related tweets and stored their tweets-id based on the corresponding trending topic, and use Google data studio to generate a fan chart. Then our front-end will use all these data to generate a website.

2.1. Streaming Session

For the twitter Streaming session, we use socket programming to bind a socket to a certain port, and use this port to get latest streaming data from twitter, and send these streaming data to our spark streaming session. We use filter in this part, in order to only retrieve the tweets we want. We also have spark program listen to this port to get these streaming data.

And in the Spark streaming session, we process the data and try to optimize our program. We count the tags that appear in the last 3 minutes and store them in Bigquery, the size of the sliding window is 1 minutes so the data in Bigquery will update every 60 seconds. And for each state, we have a corresponding table stored in Bigquery. Additionally, we use the data stored in Bigquery to obtain tweets and tweets_id about the corresponding topic and update these information every 3 minutes.

2.2. Django

For the visualization part of our project, we decide to make a website displaying all our result and the tool we used to make our website is Django. Django is a Python-based free and open-source web framework[3]. In this section we will first briefly introduce how can Django create a web App, then go into some details of the function we implemented. The final results will be showed in next section.

The fundamental workflow of our web App is as follows: we define two HTML files, result.html and detail.html. A user will firstly see the website decided by result.html whose URL is http://127.0.0.1:8000/. If he clicks one state in the map on the website, he will jump to a new address whose URL is the original URL with a suffix of the name

of the state (for example, http://127.0.0.1:8000/Illinois). On this new website defined by detail.html whose parameters are passed by the corresponding functions in views.py, detailed information will be showed to the user as well as keeping demonstrating some representative tweets. All the websites are refreshing automatically to catch up with the updated data from our Big Query while the "catch" functions belong to the back-end of Django.

The backend has three main jobs: getting the data of confirmed cases and deaths, getting the hottest hashtags in each state, and getting some tweet ids of these hashtags in each state. The confirmed cases and deaths data are stored in JHU's public Big Query and the sub-tags from our clusters are also stored in our Big Query, so we can use corresponding query codes to get these data; For the tweet id part, we use the python library tweepy whose build-in function can return the tweet ids according to the hashtag we provide. All these data will then be saved into the JavaScript file, new-us-states.js.

The JavaScript file is where the frontend get the data and it is the key for the frontend to combine all the information in a map. It has two objects: type and features. Type is the information required for combining the data to the map, and features includes 52 objects, each of which consists of the information of a US state (including the D.C. and Puerto Rico).Properties consists of information of the state, including the state name, the hottest hashtags of this state, the number of confirmed cases, the number of deaths and the tweet ids we going to display for this state. Geometry is the coordinates of the state on the global map, which help us draw the color blocks over each state map according to the confirmed cases and locate which state is a tweet belong to. The JavaScript file is loaded by the html files so these information can be displayed on the website, and the website will reload the JavaScript file every minute to refresh the data.

The main part of our website is a US map accurate to the state. When you hover your mouse over a state, an info division on the top right corner will display the information of the state, including the current time (EST), confirmed cases and deaths, the top 5 hottest hashtags.

If you click the state, a new web page with more detailed information will jump out. On top left is the administrative map of the state[2]. The source of this map for each state is only different in the state name so in views.py the state name is a parameter passed to the detail.html. Top right is some basic information of the state and a donut chart of the top 10 hottest hashtags in the state. The link to the chart is also decided by the parameter passed by views.py. The code in views.py goes as below:

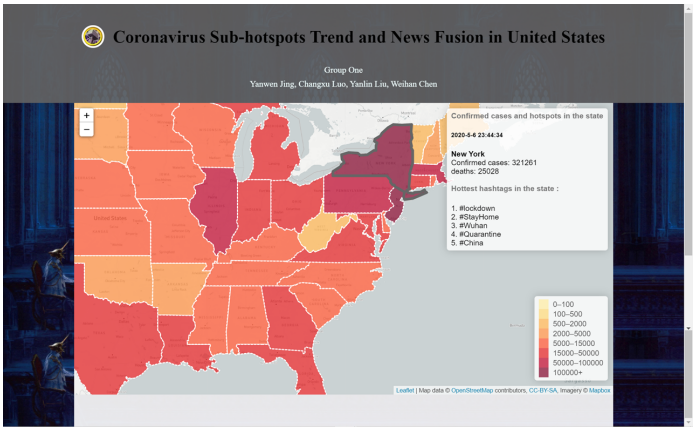

Figure 2. Hover the mouse over New York State

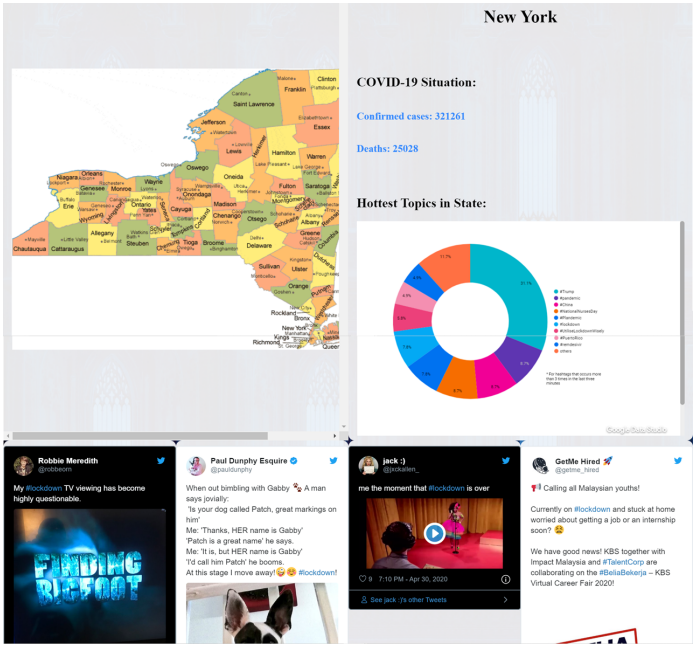

Figure 3. Detailed information page of New York State

```python
import ...
def results(request):
    context = {}
    return render(request, "result.html", context)

def getIframeLink(state_name):
    ...
    return state_chart_link

def state_details(request, state_name):
    context = {}
    context["state_detail"] = {
        "name": str(state_name),
        "hotspots": getIframeLink(state_name)
    }
```

```
16
17      return render(request, "detail.html",
        context)
```

Bottom is the tweets recommended according to the hashtags of this state. We generate a pool of tweets each time and every moment only 4 tweets are showed on the website randomly. The refresh rate of them is every 10 seconds and the refresh rate of the tweet pool is the same as the whole website.

3. Streaming Algorithms and Optimization

3.1. Streaming Algorithm

When deploying the system, there will be 52 clusters running on the cloud for every states. For each state's cluster, we maintain two jobs. The first job runs a twitter streaming session, which gets the real-time tweets streams. It filters the stream with the state's location and the tags pool in which there are a set of hashtags with the same meaning of "#covid19". It then sends the result to the second job. The second job is a Spark Streaming session, which analyzes the tweets stream, gets the trending topics, and sends the result to GCP Storage. In this part, we focus the algorithm details and optimization for the second job. The streaming pipeline DAG for the algorithm in job 2 is shown in figure 4. The program first map all the input lines with the space to get a list of all words, then it search for the symbol "#" to get the list of hashtags. After that, it removes the invalid hashtags such as "#BREAKING" and "#WATCHING" that are used by the news organizations. Then we remove the probable punctuation at the end of the hashtags, and combine all the tags with the same meaning of "#covid19", replacing them with "#covid19". This is the end of the pipeline in the first line in figure 4, and we've got a list of all the valid hashtags. We then run a map-and-reduce counting algorithm to all the hashtags, first mapping all words with 1, then performing a reduce step by adding the number, and finally sorting them by the counting number. The window duration of the counting algorithm is 24 hours, with the slide duration of 60 seconds, which means that the program will upload the results of the hashtags occurrences in the last day every minutes.

3.2. Streaming Optimization

The pipeline DAG in figure 4 has already been optimized. The optimizations mainly happens between getting the full-text tweets stream to outputting the list of valid hashtags as shown in figure 5. There are four optimizations in the algorithm. The first one is batching, which is done by the Spark DStream. It reduces the latency, and guarantees the IO safety[1]. The second one is reordering, which determines the order of operator 2 and 3. These two operators are set in this order because the selectivity of the operator 3 is less than operator 2 because the operator 3 doesn't have to remove the "invalid tags" which are not tags but words in the words list. The third optimization is the operator separation. We separate the invalid tags removing step out of the final two clearing-up steps because the selectivity of this operator 3 is less than 1. The last optimization we applied in the algorithm is to fuse the operator 4 and operator 5 into one operator, so that the program doesn't have to go through the entire words list twice, which saves the communication overhead .

To test the effect of optimization, we implement both the non-optimized version and optimized version of our algorithms as a benchmark program, and test both of them on a local machine, with the same input file which is generated locally by a virtual-tweets generation script. The input file has 500,000 pseudo tweets, each starts with a new line, and consists of 2 of the similar tags of "#covid19", 10-15 random words of 3-8 letters length. The test scripts are implemented in traditional Spark RDD instead of Dstream, so that we can measure the exact processing time in order to compare their performance differences. We run both versions on the input files for 10 times and measure the CPU running times. The results are shown in table 1.

Table 1. Result of CPU running times on non-optimized and optimized program

Round Num	Non-optimized(s)	Optimized(s)
1	13.514	9.949
2	14.903	8.802
3	14.884	8.483
4	16.396	8.609
5	14.169	8.691
6	14.080	8.631
7	14.890	8.523
8	14.606	8.074
9	14.890	8.079
10	14.260	7.982

With the result, we can calculate that, the average processing time by the non-optimized program is 14.660 seconds, while the optimized version needs 8.582 seconds in average. The improvement of the algorithm for the size of 500,000 tweets is about 41.45%, which is significant. This result tells us that these optimizations can be very useful when the input streaming is highly large-scale.

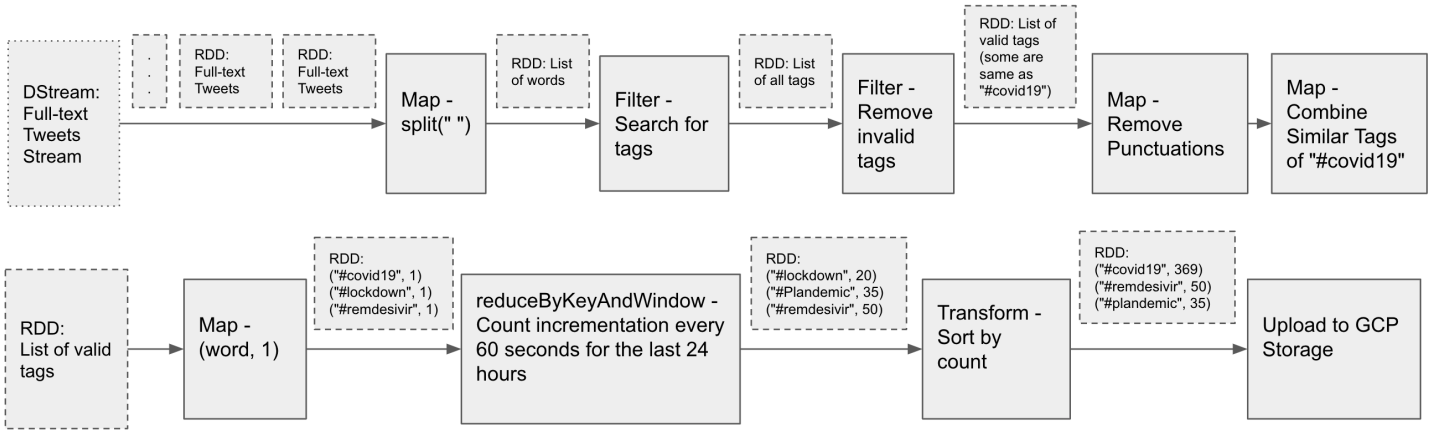

Figure 4. Streaming pipeline DAG. Please notice that the output of the pipeline in the first line is the input of the pipeline in the second line. The first line reads the tweets stream and extracts all the valid hashtags, while the second line counts them and uploads the result to the GCP Storage.

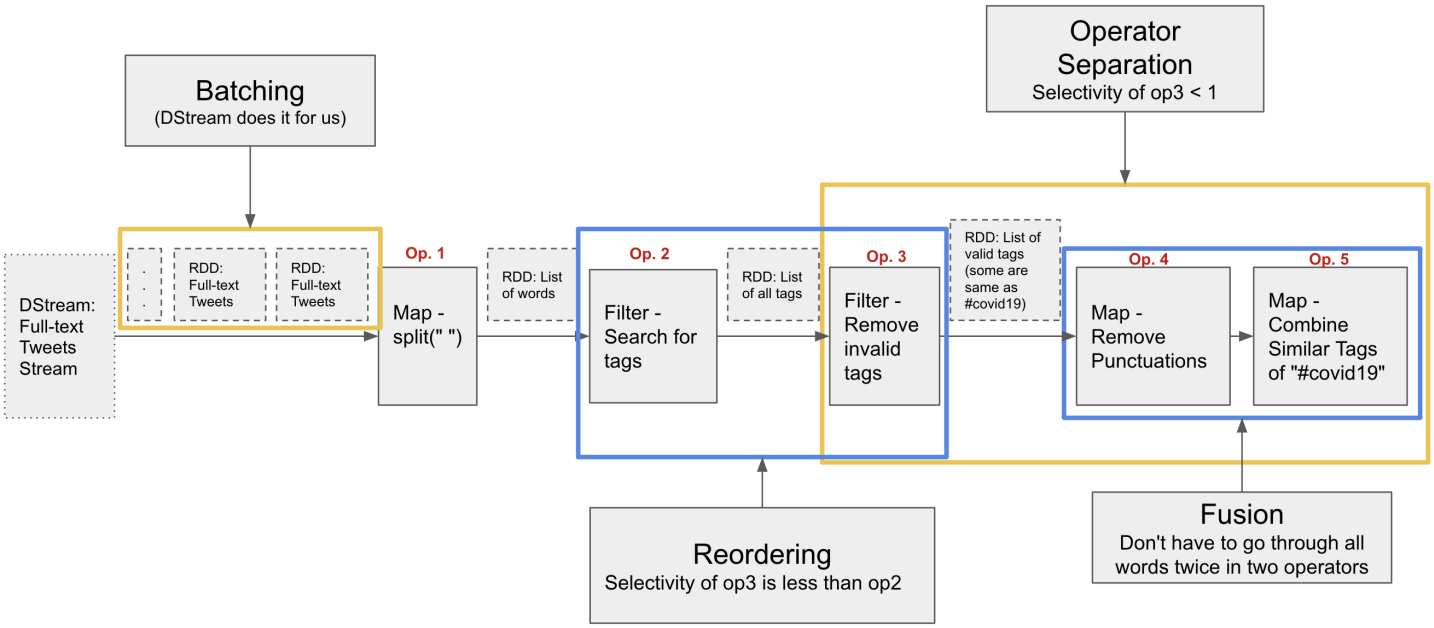

Figure 5. Optimizations to the Streaming Algorithm.

4. Demo and Results

In this part, we show some snapshots during our demo. The first part is on the Google Cloud Platform (GCP). Here we show the job details for the cluster of New York State. The output of the two jobs are shown in figure 6 and 7.

Then let's move to the front-end. The newest data from clusters has been queried so now if we hover the mouse over New York we can see the hashtags exactly the same as what we had in the second cluster, which is shown in figure 8. Click on NY, and we will jump to the detailed information page as shown in figure 9. Pay attention to the donuts chart. It clearly displays the distribution of the top 10 hottest hashtags in the state right now.

The same thing happens to all the other states, each of which has different hash tags and recommended twitters. Their data are out of date so we will not show them in the demo.

5. Conclusion and Future Work

In this project, we implemented a sub-hotspots trending and news fusion cloud application. Twitter stream is filtered by our location and hashtag filters, and hottest hashtags of each state is gotten by the Spark streaming session. These information, as well as tweets users may interested in, are

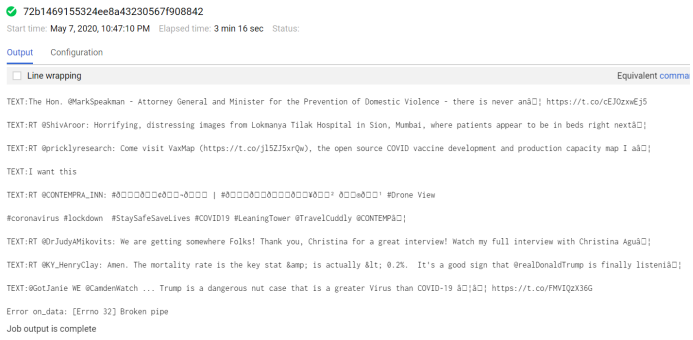

Figure 6. The first job in the cluster for New York State, showing the filtered tweets stream

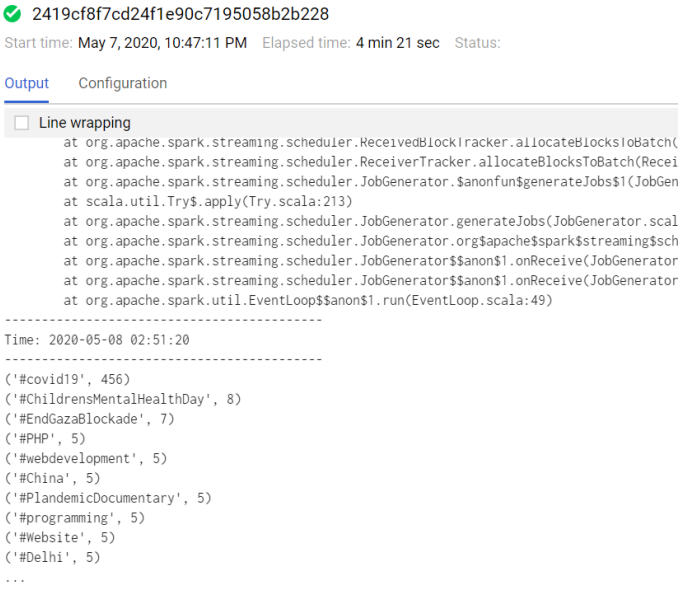

Figure 7. The second job in the cluster for New York State, showing top 10 hottest hashtags in New York State

displayed on our website. There are still some improvements we can have in the future. Firstly, we can apply semantic analysis or topic modeling to the trending sub-topics with the corresponding tweets, so that we can learn the topics in real-time and extend the presenting of topics-related information to the news literature without using hashtags. Secondly, there is more information we can try to get and fuse. We can make charts of the confirmed cases and deaths trend in past few months for each state, and twitter classification based on geographical location can be accurate to the county level. Last but not least, the front-end can be optimized. We should try combine the front-end directly to the stream so the results display can be more in time and accurate.

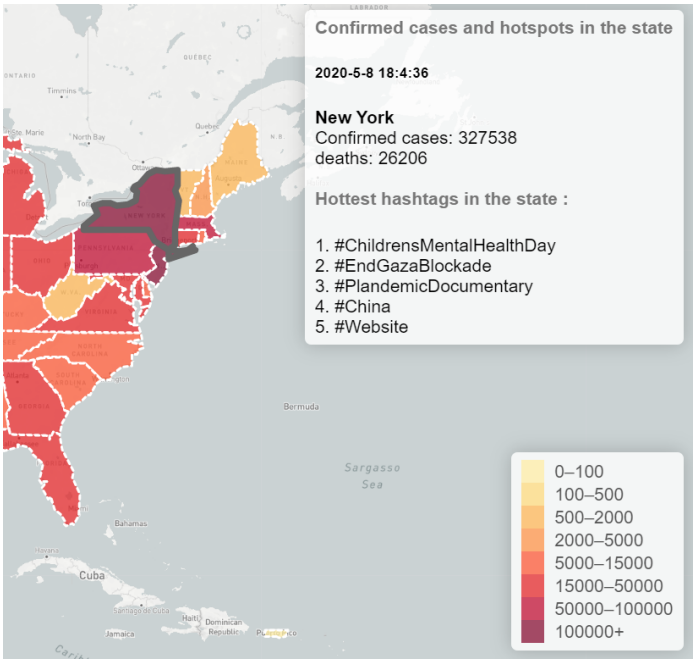

Figure 8. The main page of the website

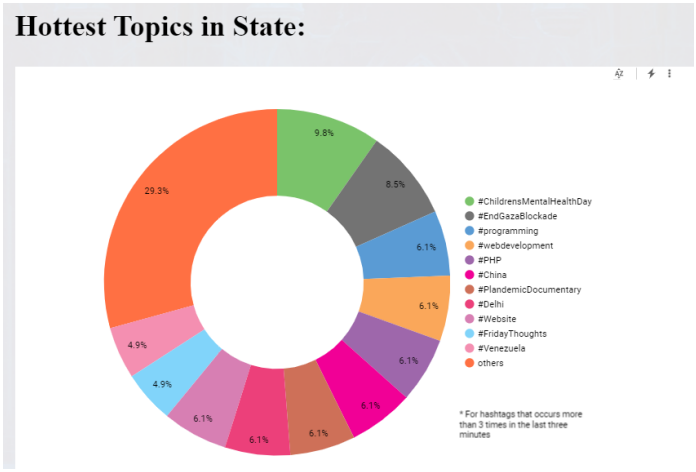

Figure 9. Donuts chart for hashtags in New York State

Acknowledgement

This work is a student project for course ELEN-6889 Large-scale Stream Processing, instructed by Professor Deepak S. Turaga. We thank him for help with this report and all useful suggestions from his feedback for the proposal and demonstration presentation.

References

[1] H. C. Andrade, B. Gedik, and D. S. Turaga. *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press, 2014.

[2] Geology. US Map Collections. `https://geology.com/state-map/`.

[3] A. Holovaty, J. Kaplan-Moss, et al. The django book. *http://www.djangobook.com/en/1.0*, 2007.

[4] Johns Hopkins University. COVID-19 Dashboard by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU). `https://coronavirus.jhu.edu/map.html`.

[5] Johns Hopkins University. COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University. `https://github.com/CSSEGISandData/COVID-19`.

[6] Twitter. Twitter Public API Documentations. `https://developer.twitter.com/en/docs`.

[7] University of Illinois. United States State Coordinates. `http://www.ala.org/rt/magirt/publicationsab/usa`.