Real-time Weather Condition Analysis

ELEN 6889 Large-scale Stream Processing

Yuhang Wang yw3733
Kangrui Li kl3350
Linyang Han lh3096
Xiaohang He xh2509

Abstract

It is not practical to search for weather information about different places in a very short time, even if keyword searching can work properly. In our project, based on this truth, we hope to design a system which displays a heatmap containing weather condition information about all the states in America by applying stream processing. We applied NOAA (National Oceanic and Atmospheric Administration) as our data source and noaa sdk API to extract the data. We utilized Spark Streaming and Django tools, and used filtering, mapping, and other streaming algorithms to deal with the data, and finally we successfully built such a real time weather condition web application demonstrating the weather information about every state.

Keywords: *Spark Streaming, Weather Condition, Real-time Analysis*

1. Problem Definition

The NOAA (National Oceanic and Atmospheric Administration) started to provide weather information among the country in 1976, offering people accurate real time weather conditions and forecasts. Nowadays people can find multiple methods to obtain weather information in their local places or other cities by using smart devices, like smartphones, watches, websites, etc. A short search for weather is very necessary for the purpose of convenience and a traditional way at present is to type the keyword of destinations on the smart device and access real time weather results and the forecast in the following hours.[2]

But an interesting thing is that people may not be able to easily search the weather in different places because they are not able to select the weather of some places they want in a very short time with those devices. So, in our project, based on this truth, we plan to design a system which could display a heat map showing the weather condition about the whole United states, and such a system should be updating frequently and very friendly and simple for people to use.

The main tasks of our project include three sections, the first would be to access weather data from some library, and we chose NOAA, which is short for National Oceanic and Atmospheric Administration, because we could access different types of weather data from this library. Then the second part is data processing, which involves raw data processing and stream processing. And in this stage, we use the knowledge we learnt from class to analyze weather data. We filter some useless data and use spark to do the selection and mapping operation. And finally, we applied Django to implement the visualization part and display results on the heat map with detailed information about weather in each state.

In the following parts, first we will describe some challenges we met during our project implementation process. Then the implementation details about data processing and visualization parts would be illustrated. Next, the streaming algorithm of our project would be presented. We will then talk about the results of our stream processing and visualization. Finally, the conclusion and future work would be discussed.

2. Challenges

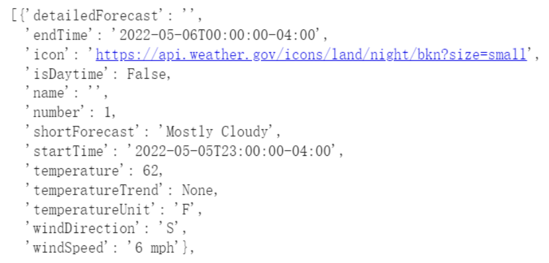
In our implementation of the project, the biggest challenge we met is the data extraction from the NOAA library. Because to obtain the weather data, we need to input the zipcodes of cities. But due to the website issue, some zip codes do not exist in the library or it may take a long time to extract the data and sometimes this operation may even fail. So we spent a lot of time selecting the correct and useful place information to get the weather data.

Another challenge would be the utilization of the Django tool because we haven't learnt basic knowledge about Java so we learnt some basic statements and how to deploy the interface system.

3. Data Streaming

3.1. Raw Data

The data required by our system is the weather data for the current day for the capitals of all states in the United States. First, we directly implement NOAA's API to fetch the raw data into our system in the format of a python list. [3] The structure of the raw data is shown below:



```
[{'detailedForecast': '',
  'endTime': '2022-05-06T00:00:00-04:00',
  'icon': 'https://api.weather.gov/icons/land/night/bkn?size=small',
  'isDaytime': False,
  'name': '',
  'number': 1,
  'shortForecast': 'Mostly Cloudy',
  'startTime': '2022-05-05T23:00:00-04:00',
  'temperature': 62,
  'temperatureTrend': None,
  'temperatureUnit': 'F',
  'windDirection': 'S',
  'windSpeed': '6 mph'},
```

Figure 1: Raw Data from NOAA

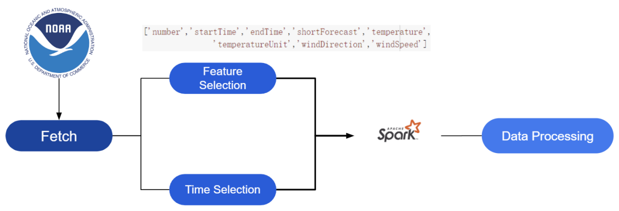
The data initially fetched is NOAA's forecast data for each area's weather for the next seven days. The data is separated by each hour. It is a list with various weather information provided by the NOAA system. But most of the information is not so useful and suitable to be presented on our weather map. So preprocessing is necessary for the following streaming.

3.2. Data Preprocessing

Using the API of NOAA, the original weather data will be fetched into our system. The raw data is the hourly weather forecast data for the next seven days. Since what our system wants to provide is the hourly updated weather information, we do not need such an excessive amount of data.

Therefore, we only retain nearly twenty-four hours of data through time selection. Also, we filter features with only selecting the raw data required for our system.

The following is the flowchart of the preprocessing procedure and some code details:



```
['number','startTime','endTime','shortForecast','temperature',
              'temperatureUnit','windDirection','windSpeed']
```

Figure 2: Data Preprocessing Procedure

The filtered data will be imported into pyspark for the subsequent processing.

3.3. Data Processing

In the data processing part, we focus more on modifying the data structure. After getting the raw data using the batch stream functions, the system runs the feature selection and time selection functions and outputs the preprocessed data into a text file. [4]

Then we use pyspark to read the file into rdd with meaningless strings. The rdd will be processed by several operators to be fed into the following streaming algorithms.We use some common operators such as map split and map replace to deal with the strings.

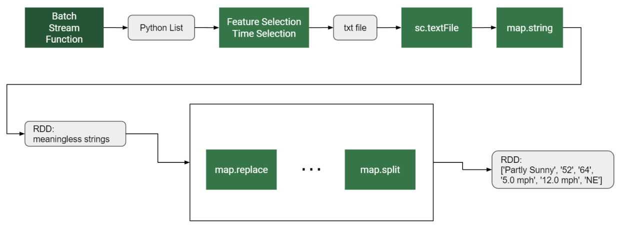


Figure 3: RDD processing

To increase the efficiency of the operators, we implemented some optimization such as separation and reordering on operators. Here is the arrangement of the operators and the implementation of the optimizations:

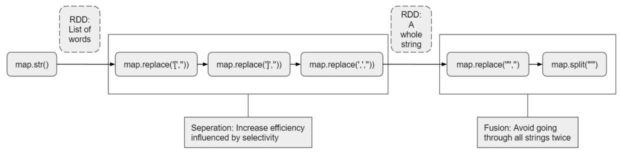


Figure 4: Streaming Optimization

3.4. Streaming Algorithm

Finally, the data will be fed into the algorithm section. We implement three different algorithms to four different groups of data. [1]

a) min / max

For the temperature data and the wind speed data, since people are most concerned about the range of these two data in their daily life, we convert them to float type and get the max and min value.

```
line.map(lambda x: x[5]) # get data
windspeed.map(lambda x: x.split(' ')[0])
windspeed.map(lambda x: float(x)) # conv
```

b) Most Frequent

For wind directions, since it can change very rapidly in many areas, we think that the most likely wind direction today might be more needed. So we assign a frequency to each predicted wind direction for each hour of the day in an area, then fetch the wind direction with the highest frequency as display data.

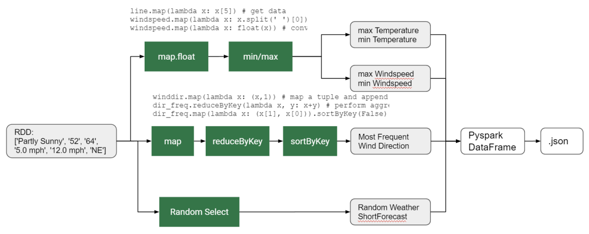
```
winddir.map(lambda x: (x,1)) # map a tuple and append
dir_freq.reduceByKey(lambda x, y: x+y) # perform aggr
dir_freq.map(lambda x: (x[1], x[0])).sortByKey(False)
```

c) Random Select

For the short forecast weather, we know that even NOAA cannot accurately predict the short-term weather in every area. So we want to do a random selection.

At first, we wanted to do a reservoir sampling. But since our current data source is hourly intervals, we only apply proportional random sampling which is relatively simple to show a short-term forecast of the weather that is likely to occur in the next 24 hours.

Since we assign an equal possibility for each data of each hour, the weather that is predicted more times is more likely to be selected. So random selection is not so random.



```
line.map(lambda x: x[5]) # get data
windspeed.map(lambda x: x.split(' ')[0])
windspeed.map(lambda x: float(x)) # conv
```

```
winddir.map(lambda x: (x,1)) # map a tuple and append
dir_freq.reduceByKey(lambda x, y: x+y) # perform aggr
dir_freq.map(lambda x: (x[1], x[0])).sortByKey(False)
```

Figure 5: Streaming Algorithms on Data

Finally, all the processed data will be written into a js file with pyspark.

3.5. Future plan

As mentioned before, what we initially wanted to build was a real-time weather visualization map. However, currently the data source from NOAA we can use is updated hourly and is not streamable. Not only did this not match the update speed of our front end, it also caused many of our optimization methods and streaming algorithms to not be available. So in this section, we will introduce some future plans about streaming if we can find a better data source.

a) Streaming Session

For the streaming session, if we have a real-time data source, we can use socket programming to bind a socket to a certain port, and use this port to get the latest streaming weather data, and send these streaming data to our spark streaming session for data processing.

In this case, some algorithms in our algorithm session can be implemented in this session to reduce the amount of useless data which will significantly decrease the running time of the whole system if we want to expand our weather map to every city of the United States. For example, we can set a filter for minimum and maximum temperature to filter out any records with the temperature value in this range. We will set a sliding window to 200 seconds to match the refresh time of our front-end webpage.

At the end of each day, the final record of the day will be stored into the system for seven days. Then the value of each column will be reset for the next day.

b) Streaming Algorithm

With the idealized real-time data source, for the random select part, we can now implement reservoir sampling to do a better random selection. The reservoir sampling is a good method to do random sampling with streaming data. The detail code is below:

```
class  Reservoir_sampling:
    def  __init__(self):
        self.result  =  None
        self.n  =  0
    def  update(self,  element):
        self.n  +=  1
        if  random()  <  1  /  self.n:
            self.result  =  element
```

The sampler can assign an equal possibility to each data in the streaming. The whole selection algorithm will automatically update with the update of the streaming data.

c) History Data

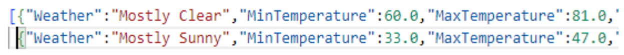
Another function we can add if we have an idealized real-time weather data source is that we can show the users the trend of the weather in the past seven days. The system will store the final record of each day in each area and keep the record for seven days. The stored data can be applied to a slider on the frontend web page. Users can swipe the bar to see weather information for each of the past seven days in each region.

Every day, the system deletes the historical data of the weather seven days ago and adds the new day's data into the database.

3.6. '.js' file

After the weather information was extracted as a format of '.json' file, the next step is to merge it into another '.json' file to combine the information of different states and weather.

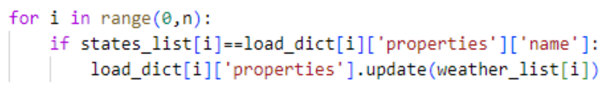
So, first, we downloaded the 'us-state.js' file from the leaflet official website and transformed it to the '.json' file, containing the geometry information of each state. Then we need to add the weather condition into corresponding states. In Figure 6, it is obvious that the weather condition is stored in a format of {key:value} pair and we can successfully add it to the state information file.



```
[{"Weather":"Mostly Clear","MinTemperature":60.0,"MaxTemperature":81.0,'
 {"Weather":"Mostly Sunny","MinTemperature":33.0,"MaxTemperature":47.0,'
```

Figure 6: Example of Weather Condition Data

Then the data of the two json files are loaded and we design the below match algorithm to make sure every weather condition information is added to the right state. And then we transformed the json file back to js file so that this can be used in the visualization part.



```
for i in range(0,n):
    if states_list[i]==load_dict[i]['properties']['name']:
        load_dict[i]['properties'].update(weather_list[i])
```

Figure 7: Match Algorithm

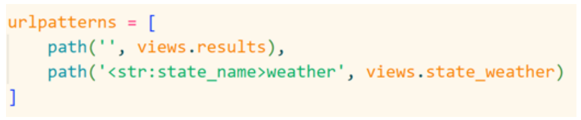
4. **Frontend Technical Description**

In this section, we will give a detailed description about how we implement our system.

4.1. Django

In the visualization step, we use Django to build our web server. Django is an open-source model driven by the high-level python programming language. It is a view controller style web application framework. We will introduce some details about our Django web App and how it is implemented. [5]

The basic working pattern of our web App is as follows: The user can go to the web server through URL 127.0.0.1:8000. In our project the server is not always-running, the user needs to run the manage.py in the code to start the server, with starting, a URL leading to the web App will also be shown as urls.py in the code files. Urls.py is as below.

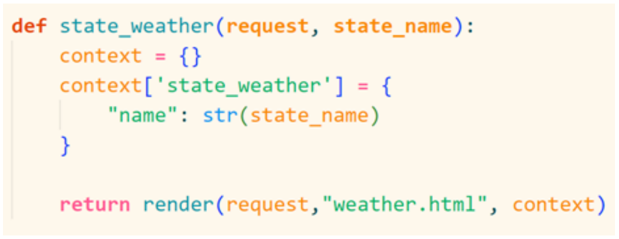


```
urlpatterns = [
    path('', views.results),
    path('<str:state_name>weather', views.state_weather)
]
```

Figure 8: Urls.py

The front-end web gets data from Javascript files and puts them on the map and interface, we can see each state as an object. In our project, the basic weather data of every state is input into Django back-end in the form of GeoJSON data which include the state(city) name, density and so on. This GeoJSON data also provides us with the shape of blocks and color of them. As these JavaScript files are loaded by the web, as html files, that information and data would be displayed on the web server.

After entering the front end web server, the user can see a map and the US states are marked with color of different density of red, this gives the user a fundamental view of the weather condition and this function is realized by views.py in code files. Furthermore, the user can click on any state of the US to get the detailed information about the state, including overall weather description, maximum and minimum temperature, maximum and minimum wind speed and the wind direction. View.py is as below.

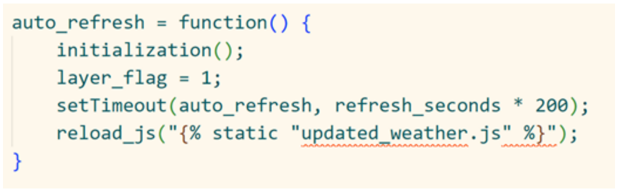


```python
def state_weather(request, state_name):
    context = {}
    context['state_weather'] = {
        "name": str(state_name)
    }

    return render(request,"weather.html", context)
```

Figure 9: View.py

The web server is refreshed every 6 seconds to connect to the streaming data from the back-end which could help to display real time data. It is realized by the auto_refresh function which uses reload_js to take new weather data from a Javascript file updated_weather circularly to renew real time streaming data to the website. Auto_refresh is as below.



```javascript
auto_refresh = function() {
    initialization();
    layer_flag = 1;
    setTimeout(auto_refresh, refresh_seconds * 200);
    reload_js("{% static "updated_weather.js" %}");
}
```

Figure 10: Auto_refresh.py

4.2. Bootstrap and CSS

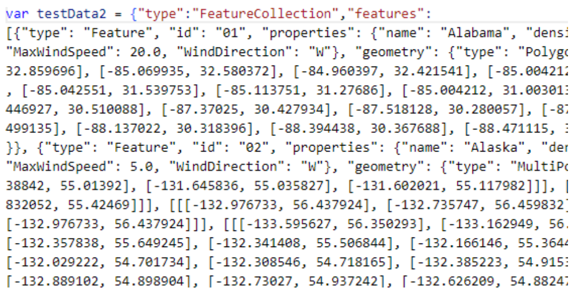
We use Bootstrap and CSS to decorate our web. Bootstrap provides a basic structure with a grid system, link style and background. It has always been a popular open-source project on GitHub. Bootstrap comes with the following features: Global CSS settings, defining basic HTML element styles, extensible classes, and an advanced grid system. We use button group, typesetting and thumbnail functions to help build the website.

4.3. Leaflet

We use Leaflet as our map display and interactive tool. Leaflet is a modern and open-source JavaScript library developed for building mobile device friendly interactive maps. Leaflet only weighs just about 39 KB of JS, and it has all the mapping features most developers ever need. In our project, we are using mapbox's api. [6]

5. Results

For the results part, we would display some snapshots of processed data and user interface as shown in the demo slot. Here, first we present the processed data below in Figure 11 and this is some information in the '.js' file before being used in Django.



```
var testData2 = {"type":"FeatureCollection","features":
[{"type": "Feature", "id": "01", "properties": {"name": "Alabama", "dens
"MaxWindSpeed": 20.0, "WindDirection": "W"}, "geometry": {"type": "Polyg
32.859696], [-85.069935, 32.580372], [-84.960397, 32.421541], [-85.00421
, [-85.042551, 31.539753], [-85.113751, 31.27686], [-85.004212, 31.00301
446927, 30.510088], [-87.37025, 30.427934], [-87.518128, 30.280057], [-8
499135], [-88.137022, 30.318396], [-88.394438, 30.367688], [-88.471115,
}}, {"type": "Feature", "id": "02", "properties": {"name": "Alaska", "de
"MaxWindSpeed": 5.0, "WindDirection": "W"}, "geometry": {"type": "MultiP
38842, 55.01392], [-131.645836, 55.035827], [-131.602021, 55.117982]]],
832052, 55.42469]]], [[[-132.976733, 56.437924], [-132.735747, 56.459832
[-132.976733, 56.437924]]], [[[-133.595627, 56.350293], [-133.162949, 56
[-132.357838, 55.649245], [-132.341408, 55.506844], [-132.166146, 55.364
[-132.029222, 54.701734], [-132.308546, 54.718165], [-132.385223, 54.915
[-132.889102, 54.898904], [-132.73027, 54.937242], [-132.626209, 54.8824
```

Figure 11: weather information in '.js' file

For the frontend website, we provide the search results that users can obtain with our heat map here. After navigating to the URL, you can see the basic interface of the weather system as in Figure 12. The user can view the overall weather conditions that are different from state to state. The states show different density of red color, the deeper the red color is means the higher the maximum temperature of the state. The level of color, referring to temperature, is shown at the right lower corner. The right upper corner includes the current time of the system, the time and streaming data will refresh every 6 seconds.

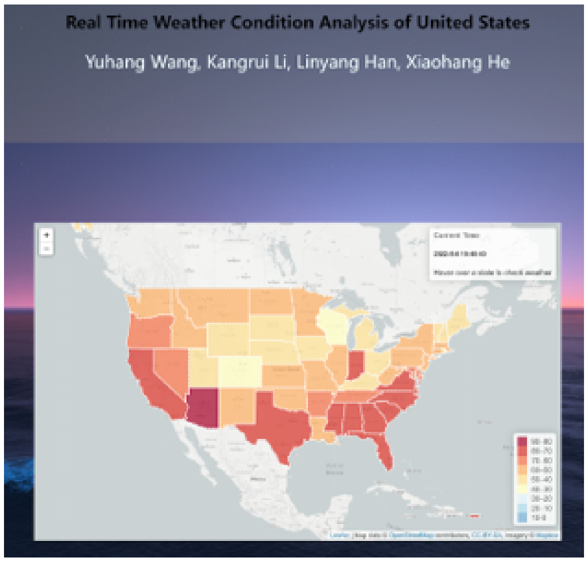

Real Time Weather Condition Analysis of United States
Yuhang Wang, Kangrui Li, Linyang Han, Xiaohang He
Current Time
Hover over a state to check weather

Figure 12: Overview of Basic Interface

In the heat map, the newly updated weather information in each state is displayed, including max and min temperature, weather conditions. Take New York as an example, once you move to its section, some general information including weather condition, min and max temperature will be displayed like in Figure 13.

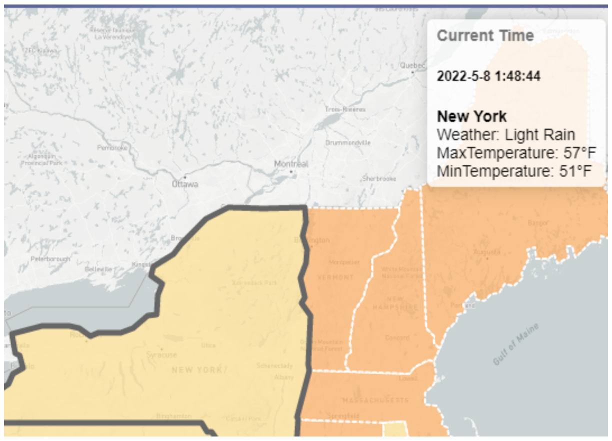
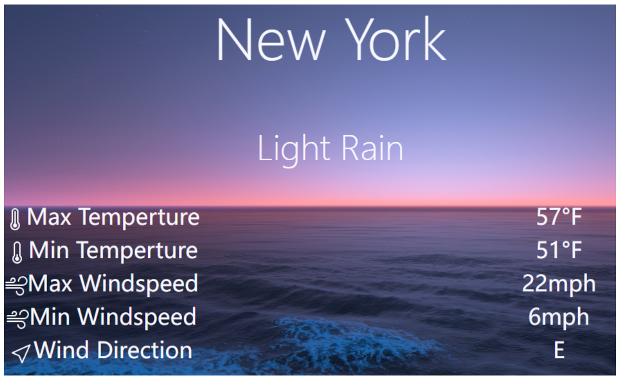


Figure 13: Main page of New York Weather

The user can then click into any state to see detailed weather conditions of the state as Figure 14. The overall weather description, ‘Light Rain’ in Figure 14, followed with maximum and minimum temperature, maximum and minimum wind speed and wind direction. Users could go back to the map interface to view other states’ conditions afterwards.



New York

Light Rain

Max Temperture	57°F
Min Temperture	51°F
Max Windspeed	22mph
Min Windspeed	6mph
Wind Direction	E

Figure 14: Detailed information in New York

6. Future Work

Although we successfully built a weather analysis system in our project, there is still some work we can do in the future to improve our project. Firstly, we could expand the size of the dataset and try to cover the weather data in every city in every state, which could make our system more complete and meet the search needs of people in different cities. Also, in the aspect of expanding data, we could involve more weather parameters like humidity, air pressure, rain amount, etc.

The second job we can improve our system is that we could involve streaming optimization analysis to our streaming algorithm. We can apply some optimization techniques, like reordering, redundancy elimination and load shedding. By applying the optimization, the speed of data processing could be increased a lot.

And the last improvement we can make to our project is to find a better data source of weather. Although the NOAA weather library is pretty good, containing lots of weather information on different types, the weather information could only be updated once an hour, so, we will try to find a replacement for the data to make the data update more frequently.

Acknowledgements

This report is a detailed explanation of our project for course ELEN 6889 Large-scale Stream Processing. We want to thank our instructor of this course, dear professor Deepak S. Turaga, who has given us lots of useful suggestions and feedback for our project and report writeup. Also, we want to thank him for all the advanced knowledge related to stream processing presented by him in the lecture.

References

[1] C. Andrade, B. Gedik, and D. S. Turaga. Fundamentals of stream processing: application design, systems, and analytics. Cambridge University Press, 2014.
[2] NOAA. NOAA Weather Forecasts Documentation. https://www.weather.gov/forecastmaps.
[3] NOAA SDK. https://pypi.org/project/noaa-sdk/.
[4] Spark.https://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-operations
[5] A. Holovaty, J. Kaplan-Moss, et al. The django book. http://www.djangobook.com/en/1.0 , 2007.
[6] Leaflet. Interactive Choropleth Map - Leaflet - a JavaScript library for interactive maps (leafletjs.com)