ELEN E6889 - Homework 1

Tong Wu, tw2906

Introduction

This homework uses python and pyspark to process some large scale data. This homework is formed by four parts:

1. Read the web server log, provide the summarized bytes by each individual IP. Then, output a csv file that contains result which is ordered by IP in string ascending order.
2. Provide IPs that served most number of bytes. Then, output two csv files that contains top 10 and top 100 of tuple <IP, bytes>, ordered in descending order of bytes.
3. Compute the total number of bytes served per time window for 1 hour for each unique IP, with tumbling window. Then output a csv file contains result from time window `30:00:00:00` to `30:00:59:59` , sort by IP in string ascending order.
4. Provide the summarized bytes by each subnets `xxx.xxx.xxx.*` . Then, output a csv file contains the tuple of <subnet, bytes>, where the subnet here is `xxx.xxx.*.*` , sort by subnet in string ascending order.

Implementation

Part 1

- Use `sc.textFile("path")` to read the txt file. Found that the file is separated by space with tuple <IP/domain, time_stamp, http_command, response_code, bytes>, so split the file with space using function `map(lambda x: x.split(' '))` in order to process it smoothly.
- Extract the tuple <IP, bytes> using `map()` .
- Summarize all columns according to the IP, using `reduceByKey()` .
- Create a function “is_ip_address” in odrder to identify whether the input is an IP address or not, filter out all non-IP in the first row by using `filter(lambda x: is_ip_address(str(x[0])))` . Then the result is outputted in the screen.
- Use `spark.createDataFrame()` to convert the pipelined RDD data type into dataframe. Then, use `coalesce(1).write.csv()` to output the result into a csv file.

Part 2

- Based on part 1, using the filtered output from part 1 to continue this part.
- Use `sortBy(lambda x: x[1], False)` to sorting the data according to the specific row. The option `False` means that sorting in descending, and `True` for ascending.
- Use `take(10)` or `take(100)` to form top 10 and top 100 of the data. Then use the same way in part 1 to output the csv files.

Part 3

- Extract time from the original data, only need day and hour from time tuple DD:HH:MM:SS.
- Split IP, date, hour and bytes into separate rows. Use `replace(':', ' ')` to separate the date and hour. Extract IP and bytes is discussed in part 1.
- Use "is_ip_address" function to filter out the non-ip address.
- In order to compute the bytes in 1 hour time window, use `filter(lambda x: x[1]=day and x[2]=hour)` to filter out the specific day and hour. Use for loop to group all time windows.
- In the for loop, use `map()` to extract IP and bytes to form tuple. Then summarize bytes up according to IP using `reduceByKey()`
- In order to output the result in time window `30:00:00:00` to `30:00:59:59` , use `filter()` to filter out the specific window, then use `sortBy()` and `map()` to form a ascending tuple.
- Output the result by using the same way in part 1.

Part 4

- Use the filtered data from part 1 to continue this part.
- In order to delete the last part or the last two parts of the IP address, first change the data type to dataframe, then use `map()` and `split()` to extract the subnet. Then summarize by subnet using `reduceByKey()` , and sort by subnet in string ascending order using `sortBy()`
- Output the result by using the same way in part 1.

Result

Below are results that show in the terminal.

Part 1

`filtered_output`

```
[('128.104.66.114', 4889),
 ('128.120.125.246', 11359),
 ('128.120.153.224', 480433),
 ('128.120.5.70', 0),
 ('128.120.68.34', 61554),
 ('128.138.177.19', 156204),
 ('128.138.177.44', 167591),
 ('128.138.177.6', 63856),
 ('128.144.102.56', 521757),
 ('128.148.245.158', 3947)]
```

Part 2

`top_10`

```
[('139.121.98.45', 5041738),
 ('203.251.228.110', 3785626),
 ('155.84.92.3', 3353172),
 ('198.102.67.27', 3182052),
 ('161.122.12.78', 3008684),
 ('156.98.205.46', 1938034),
 ('141.243.1.172', 1634402),
 ('204.7.162.158', 1463223),
 ('129.237.24.71', 1274796),
 ('137.219.55.101', 1162480)]
```

`top_100`

```
[('139.121.98.45', 5041738),
 ('203.251.228.110', 3785626),
 ('155.84.92.3', 3353172),
 ('198.102.67.27', 3182052),
 ('161.122.12.78', 3008684),
 ('156.98.205.46', 1938034),
 ('141.243.1.172', 1634402),
 ('204.7.162.158', 1463223),
 ('129.237.24.71', 1274796),
 ('137.219.55.101', 1162480),
 ('146.138.145.184', 1118189),
 ('146.138.145.206', 1098374),
 ('146.138.34.168', 1079474),
 ('128.159.128.27', 1052103),
 ('130.19.10.65', 957522),
 ('156.41.19.11', 798931),
 ('198.106.169.52', 788749),
 ('131.167.25.3', 784339),
 ('204.130.16.23', 758336),
 ('163.234.224.24', 604773),
 ('149.168.51.35', 531611),
 ...
```

Part 3

Tumbling window:

```
29H:23H
[('141.243.1.172', 1497), ('140.112.68.165', 7811), ('131.215.67.47', 32988), ('131.170.154.29', 7513)]

30D:0H
```

```
[('161.122.12.78', 45957), ('137.132.52.66', 2235), ('141.243.1.174', 10739), ('141.243.1.172', 177370), ('138.25.148.25', 28591), ('204.188.47.212', 21118), ('131.170.154.29', 5075), ('202.32.50.6', 16682), ('149.159.22.10', 67951), ('157.22.192.30', 7513), ('204.62.245.32', 21181), ('203.1.203.222', 3717)]

30D:1H
[('161.122.12.78', 5505), ('138.25.148.25', 16900), ('140.120.29.161', 11516), ('203.1.203.111', 5686), ('203.1.203.104', 5686), ('141.243.1.172', 20750), ('203.1.203.120', 5686), ('204.62.245.32', 8893), ('131.203.13.244', 4889), ('198.69.241.75', 34462), ('202.96.29.111', 10296)]

30D:2H
[('141.243.1.172', 352256), ('141.192.85.41', 13870), ('161.122.12.78', 876514), ('131.181.38.71', 18480), ('202.32.50.6', 60604), ('204.62.245.32', 24710), ('168.95.125.161', 156621), ('149.250.222.1', 4254)]

30D:3H
...
```

Part 4

`subnet xxx.xxx.xxx.*`

```
[('140.112.68', 7811),
 ('202.32.50', 159922),
 ('149.159.22', 67951),
 ('198.69.241', 34462),
 ('202.96.29', 10296),
 ('168.95.125', 156621),
 ('141.192.85', 43782),
 ('149.250.222', 4254),
 ('203.251.228', 3785626),
 ('132.74.12', 69622)]
```

CSV file

The outputted csv file is in the folder `output`