Stream Processing 2nd Project

João Martins 52422, Ricardo Ferreira 52915

Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

1 Introduction

This project aims to present solutions (queries) on a dataset that contains information about taxi rides in New York City. Using Siddhi, which allows creating Apps where we can write Streaming processing and Complex Event Processing rules with Siddhi Streaming SQL, and WSO2 Streaming Processor which allows to integrate Siddhi Apps, we intend to present solutions for the following questions:

- **Q1:** Find the top 10 most frequent routes during the last 30 minutes.
- **Q2:** Identify areas that are currently most profitable for taxi drivers.
- **Q3:** Alert whenever the average idle time of taxis is greater than a given amount of time (say 10 minutes).
- **Q4:** Detect congested areas.
- **Q5:** Select the most pleasant taxi drivers.

To divide the map into an NxN grid, given the longitude and latitude values of the event, we use the following formulas to calculate the grid cell:

$$LONG\_CELL = round((longitude - MIN\_LON)/LON\_DELTA) \quad (1)$$

$$LAT\_CELL = round((MAX\_LAT - latitude)/LAT\_DELTA) \quad (2)$$

where $MIN\_LON = -74.916578$, $MAX\_LAT = 41.47718278$ and LON_DELTA and LAT_DELTA are deltas corresponding to the intended N for the NxN grid.

This report is structured as follows: in section 2 we present the rationale for solving each issue as well as our interpretation of the issue; in section 3, we present in greater detail our solution to one of the questions, as well as the results obtained; and, finally, in section 4, we present our conclusions about the solutions and results obtained.

2 Queries Presentation

In this section we individually present the interpretation and rationale for our solutions.

2.1 Question 1

Question 1 is intended to: *"Find the top 10 most frequent routes during the last 30 minutes"*.

This particular query is the same that was asked in the first delivery of this course and the execution was inspired in that same implementation with spark streaming.

At first we started by translating the coordinates to a 300x300 grid as requested in the question. After that we filter all the invalid coordinates of the new grid since all valid values should be between 0 and 300.

In order to get the top 10 most frequent areas we run a query which selects the most common coordinate combinations, of pickup and drop off, and their *count()*, grouped by all the coordinates of a ride, orders them in descending order, of the *count()*, and limit the top to 10 results.

2.2 Question 2

Question 2 is intended to: *"Identify areas that are currently most profitable for taxi drivers"*.

For this, we start by dividing the map into a 600x600 grid. To identify the most profitable areas, we divided the profit made in that area by the number of empty taxis. For each area we calculate the average profit in that area, where we consider the sum of fares plus the sum tips (using the aggregation function *sum*) dividing by the number of trips (using the aggregation function *count*), in each area in the last 15 minutes.

To calculate the number of empty taxis, we selected, for each taxi, its highest dropoff_datetime, in the last 30 minutes, and for each area we count the taxis, using the aggregation function *count*.

For the presentation of results, we sorted by average profit, descending, selecting only the first five results.

2.3 Question 3

Question 3 is intended to: *"send an alert whenever the average idle time of taxis is greater than 10 minutes"*.

In our implementation we start by only selecting the medallion, to identify the taxi, and the pickup and drop off times for each event. In this selection we perform a parsing of the string value of the times to a timestamp using "$time : timestampInMilliseconds(time\_string, format)$", for example.

Next we use the "arrow" (->) logical pattern in order to select the sequence of drop off and pickup times from taxis with the same medallion and use it to calculate the idle time between rides of the same taxi.

After that we perform a query over a time batch of 10 seconds so that we can see the results quickly and we select the computed average idle time, calculated with the function $avg()$ and we add a $having\ avg\_idle\_time > (10.0 * 60.0 * 1000.0)$ so that we only alert when the average idle time exceeds 10 minutes.

At last at the sink we provide an alert message which says that a 10 minute threshold was exceeded.

2.4 Question 4

Question 4 is intended to: *"Detect congested areas"*.

For this query, as in query 2 we split the map into a 600x600 grid. To identify the most congested areas, we used a logical pattern, selecting the areas that had a sequence of four trips, all increasing in duration, accessing the values in *trip_time_in_secs*.

More details about this query in the Detailed Query Presentation section.

2.5 Question 5

Question 5 is intended to: *"Select the most pleasant taxi drivers"*.

For this query, the solution was quite simple. To select the most pleasant taxi driver, we grouped by taxi driver, adding (using the aggregation function *sum*) the tips received by the taxi driver on all trips that occurred in the time window. Then, sorting by the tip value in a descending way, just select the taxi driver that is at the top, using "*limit 1*".

3 Detailed Query Presentation

In this section we present in more detail one of the answered questions, presenting the code and the rationale behind the decisions taken. The question we are going to present is Q4: *"Detect congested areas"*.

```
----------
@source(type='kafka',
        topic.list='productions',
        partition.no.list='0',
        threading.option='single.thread',
        group.id="group",
        bootstrap.servers='kafka:9092',
        @map(type='csv', delimeter=','))
define stream RawInputStream (medallion string,hack_license string,
        pickup_datetime string, dropoff_datetime string,
        trip_time_in_secs long,trip_distance string,
        pickup_longitude double,pickup_latitude double,
        dropoff_longitude double, dropoff_latitude double,
        payment_type string,fare_amount double,
        surcharge string,mta_tax string,tip_amount double,
        tolls_amount double,total_amount double);

define stream GridInputStream (medallion string, pick_lon_grid long,
        pick_lat_grid long, drop_lon_grid long,
        drop_lat_grid long, trip_time_in_secs long);
```

```
define stream FilteredGridInputStream (medallion string, pick_lon_grid long,
        pick_lat_grid long, drop_lon_grid long,
        drop_lat_grid long, trip_time_in_secs long);

@sink(type='log')
define stream CongestedAreasStream (pick_lon_grid long, pick_lat_grid long);
----------
```

For this question we use 4 streams to process the data, where RawInputStream is the stream that takes the data directly from the dataset's csv and CongestedAreasStream is the output stream where it gives us the answers of which are the most congested areas.

```
----------
@info(name='query2')
from RawInputStream
select medallion, math:round((pickup_longitude - (-74.916578)) /
(0.005986 / 2)) as pick_lon_grid,
math:round((41.47718278 - pickup_latitude) / (0.004491556 / 2)) as pick_lat_grid,
math:round((dropoff_longitude - (-74.916578)) / (0.005986 / 2)) as drop_lon_grid,
math:round((41.47718278 - dropoff_latitude) / (0.004491556 / 2)) as drop_lat_grid,
trip_time_in_secs
insert into GridInputStream;
----------
```

Using the formulas presented above, we transformed the longitude and latitude coordinates into grid cells, thus selecting the trip pickup and dropoff grid cells and also the medallion and the trip time (*trip_time_in_secs*), inserting the data in the GridInputStream stream.

```
----------
@info(name='query3')
from GridInputStream [ pick_lon_grid >= 0 and pick_lon_grid <= 600
        and pick_lat_grid >= 0 and pick_lat_grid <= 600 and
        drop_lon_grid >= 0 and drop_lon_grid <= 600 and
        drop_lat_grid >= 0 and drop_lat_grid <= 600 ]
select medallion, pick_lon_grid, pick_lat_grid,
        drop_lon_grid, drop_lat_grid, trip_time_in_secs
insert into FilteredGridInputStream;
----------
```

Then we filter only the events whose dropoffs and pickups occur with a latitude cell greater than zero and less than 600 and a longitude cell greater than zero and less than 600, inserting the same attributes of GridInputStream in FilteredGridInputStream.

```
----------
@info(name='query7')
from every e1=FilteredGridInputStream
    -> e2=FilteredGridInputStream[e1.pick_lon_grid == e2.pick_lon_grid and
                                  e1.pick_lat_grid == e2.pick_lat_grid and
                                  e2.trip_time_in_secs > e1.trip_time_in_secs]
    -> e3=FilteredGridInputStream[e2.pick_lon_grid == e3.pick_lon_grid and
```

```
                                        e2.pick_lat_grid == e3.pick_lat_grid and
                                        e3.trip_time_in_secs > e2.trip_time_in_secs]
    -> e4=FilteredGridInputStream[e3.pick_lon_grid == e4.pick_lon_grid and
                                        e3.pick_lat_grid == e4.pick_lat_grid and
                                        e4.trip_time_in_secs > e3.trip_time_in_secs]
    within 10 sec
select e1.pick_lon_grid, e1.pick_lat_grid
insert into CongestedAreasStream;
----------
```

Finally, over the last 10 seconds, we used a logical pattern, selecting the areas that had a sequence of four trips, all increasing in duration, accessing the values in *trip_time_in_secs*. In this way we identify areas that have had a peak in travel time and are therefore the most congested areas.

We insert the results into the *CongestedAreasStream*, which is the output stream that prints the results to the console (as it was defined with *sink(type='log')*)

4 Conclusion

conclusion