Big Data in NYC for Taxi rides management

with Apache Hive

Gabriele Favia matr. 579166

December 2019

Contents

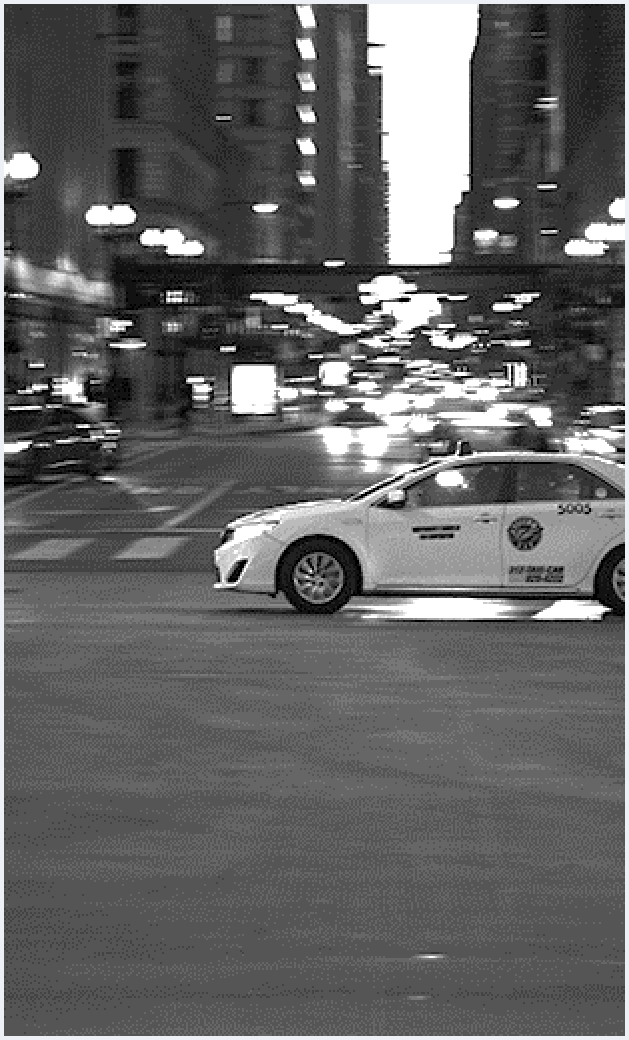
The NYC Taxi and Limousine Commission

The New York City Taxi and Limousine Commission (NYC TLC) is an agency of the New York City government that licenses and regulates the medallion taxis and for-hire vehicle industries, including app-based companies.
The TLC's regulatory landscape includes medallion (yellow) taxicabs, green or Boro taxicabs, black cars (including both traditional and app-based services), community-based livery cars, commuter vans, paratransit vehicles (ambulettes), and some luxury limousines [1].

But what is the difference between yellow and green taxi (cab in American English)?

The famous NYC yellow taxis provide transportation exclusively through street-hails.
The number of taxicabs is limited by a finite number of medallions issued by the TLC.
It’s possible to access this mode of transportation by standing in the street and hailing an available taxi with your hand. The pickups are not pre-arranged.

The green taxis know as well as “Boro” cab or Street Hail Livery (SHL) are permitted to accept street-hails above 110th Street in Manhattan and in the outer-boroughs of New York City. The SHL program allows livery vehicle owners to license and outfit their vehicles with green borough taxi branding, meters, credit card machines, and ultimately the right to accept street hails in addition to pre-arranged rides [2].



Requests

Designing and implement in Hadoop MapReduce:

4. A HiveQL query returning the total number of passengers in yellow and green taxi in 2018.

5. A HiveQL query returning, for each rate code, the total amount charged to passengers of yellow and green taxi in 2018.

6. A HiveQL query returning, for each *PULocationID*, the list of related *DOLocationID* for yellow taxi in 2018, ordered by increasing average trip distance.

Data dictionary

The two kind of file of interest, yellow and green have a different header, which means that the software has the need to distinguish and operate the two cases differently.

Moreover, the green and yellow taxis dataset produced from January 2016 to June 2016, follow a different data dictionary that requires some variation to the parsing procedure for the points 4 and 5, while for the point 6, the dataset cannot be use because of the absence of these fields.

The fields of interest to be extracted from both the yellow and green cabs files are:

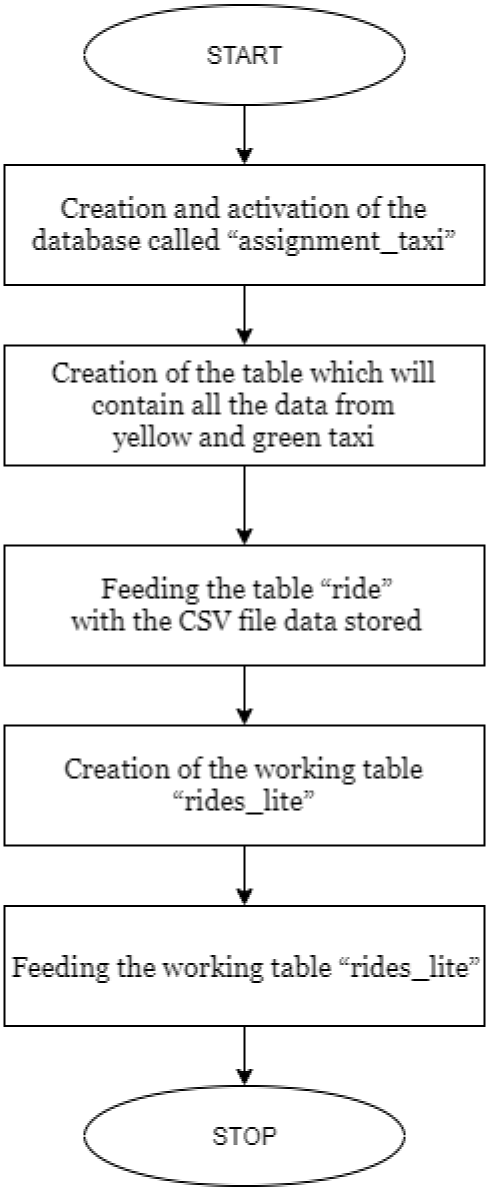
Field	Description
lpep_pickup_datetime or rpep_pickup_datetime	The pickup time, useful to get the year.
Passenger_count	The number of passengers in the vehicle.
RatecodeID	The final rate code in effect at the end of the trip. 1 = Standard rate 2 = JFK 3 = Newark 4 = Nassau or Westchester 5 = Negotiated fare 6 = Group ride
total_amount	The total amount charged to passengers. Does not include cash tips.

While the fields of interest exclusively extracted from yellow files are:

Field	Description
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
trip_distance	The elapsed trip distance in miles reported by the taximeter.

A comprehensive description of all the fields is available at nyc.gov website both for yellow [3] and the green [4] cabs.

DDL and DML commands


START
Creation and activation of the database called “assignment_taxi”
Creation of the table which will contain all the data from yellow and green taxi
Feeding the table “ride” with the CSV file data stored
Creation of the working table “rides_lite”
Feeding the working table “rides_lite”
STOP

Creation and activation of the database called “assignment_taxi”

```
CREATE DATABASE IF NOT EXISTS assignment_taxi;
USE assignment_taxi;
```

Creation of the table which will contain all the data from yellow and green taxi

```
CREATE TABLE rides (
VendorID INT,
lpep_pickup_datetime STRING,
lpep_dropoff_datetime STRING,
store_and_fwd_flag STRING,
RatecodeID INT,
PULocationID INT,
DOLocationID INT,
passenger_count INT,
trip_distance FLOAT,
fare_amount INT,
extra INT,
mta_tax INT,
tip_amount INT,
tolls_amount INT,
ehail_fee STRING,
improvement_surcharge INT,
total_amount INT,
payment_type INT,
trip_type INT
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' tblproperties
("skip.header.line.count"="2");
```

- `ROW FORMAT DELIMITED FIELDS TERMINATED BY ','` sets the delimiter of the columns to be a comma.
- `tblproperties ("skip.header.line.count"="2")` makes the process of data importing skipping the first two rows of the CSV document, which are made up of a header and a blank row.

Feeding the table “ride” with the CSV file data stored

```
LOAD DATA LOCAL INPATH '/media/sf_condiv/Hive_7Dec/input' OVERWRITE INTO
TABLE rides;
```

Creation of the working table "rides_lite"

To make the queries faster and reducing the memory usage at the same time, a smaller version of the loaded table is created, containing only the fields useful for the queries (the ones described in Data dictionary section).

```
CREATE TABLE rides_lite (
year INT,
RatecodeID INT,
PULocationID INT,
DOLocationID INT,
passenger_count INT,
trip_distance FLOAT,
total_amount INT,
taxi_type INT
);
```

- The `taxi_type` field is added with the aim of distinguishing with ease (just a INT flag) whether the row of interest is relative to a green (value 1) or a yellow cab (value 2).

Feeding the working table "rides_lite"

```
INSERT OVERWRITE TABLE rides_lite
SELECT
YEAR(lpep_pickup_datetime) as year,
RatecodeID,
PULocationID,
DOLocationID,
passenger_count,
trip_distance,
total_amount,
CASE WHEN (REVERSE(SPLIT(REVERSE(INPUT__FILE__NAME), '[/]')[0])) LIKE
'green%' THEN 1 WHEN (REVERSE(SPLIT(REVERSE(INPUT__FILE__NAME),
'[/]')[0])) LIKE 'yellow%' THEN 2 END
FROM rides;
```

- Noticing that the objective queries require only the year of a ride, it's been chosen to insert only this inside the `rides_lite` table using the function `YEAR` which extracts the year from a string date. This expedient lightens the computational and memory requirements, also because the year is here treated as an integer instead of a string.
- To value the field `taxi_type` of the table `rides_lite`, Hive checks if the file name starts with the keyword green or yellow:
  The keyword `INPUT__FILE__NAME` is a virtual column [5] which contains the full path of the source file related to that row.
  The algorithm consists of reversing the full path and getting the first word delimited by a slash (resulting to be the file name with reversed letters disposition), extracting the first element and reversing it again to get the file name of the CSV file; then, depending by the type of the taxi, Hive assigns 1 or 2 to the `taxi_type` field using the `CASE` statement and the `LIKE` operator.

INPUT__FILE__NAME	hdfs://localhost:9000/user/hive/warehouse/assignment_taxi.db/rides/green_tripdata_2016-07.csv
REVERSE	vsc.70-6102_atadpirt_neerg/sedir/bd.ixat_tnemngissa/esuoheraw/evih/resu/0009:tsohlacol//:sfdh
SPLIT on /	[0 => vsc.70-6102_atadpirt_neerg] [1 => sedir] [2 => bd.ixat_tnemngissa] [3 => esuoheraw] [4 => evih] [5 => resu] [6 => 0009:tsohlacol] [7 =>] [8 => :sfdh]
SPLIT_ELEMENT[0]	vsc.70-6102_atadpirt_neerg
REVERSE	green_tripdata_2016-07.csv

DQL commands

To store the result in a file it is possible to put before the query, the following command:

```
INSERT OVERWRITE LOCAL DIRECTORY 'home/result_hive'
```

Query 4

```
SELECT year, SUM(passenger_count) FROM rides_lite GROUP BY year HAVING
year = '2018';
```

- This query uses the SUM function to sum the *passenger_count* and group the results by year.

Query 5

```
SELECT RatecodeID, SUM(total_amount) FROM rides_lite WHERE year = '2018'
GROUP BY RatecodeID;
```

This query uses the SUM function to sum the *total_amount*, for rides in 2018 and group the results by RatecodeID.

Query 6

```
SELECT PULocationID, COLLECT_SET(DOLocationID) as group_doloc,
COLLECT_LIST(avg_trip) as group_trip FROM (SELECT PULocationID,
DOLocationID, CAST(FORMAT_NUMBER(AVG(trip_distance), 2) AS FLOAT) AS
avg_trip FROM rides_lite WHERE taxi_type = 1 AND year = '2018' GROUP BY
PULocationID, DOLocationID ORDER BY avg_trip) table_temp GROUP BY
PULocationID;
```

This query is made up of two queries:

- Internal query: selecting all the yellow taxi (`taxi_type = 1`), it gets the values of the columns *PULocationID* and *DOLocationID* and groups the average of *trip_distance* by *PULocationID*. The average results are expressed in miles, approximated to the second decimal digit, for ease of reading purposes. The results are sort by average trip.
- External query: groups by *PULocationID* and display the results of *DOLocationID* and *avg_trip* in a “list-like” mode.

Dealing with different datasets

To make the measurement comparison three datasets have been created:

- A table containing data for 2018, 2017 e the second half of 2016, inside the table *rides_lite*.
- A table containing data for 2018 and 2017 only, called *rides_lite_2018_2017*.
- A table containing data from 2018 only, called *rides_lite_2018*.

Construction and population of the table for the years 2018 and 2017

```
CREATE TABLE rides_lite_2018_2017 (
year INT,
RatecodeID INT,
PULocationID INT,
DOLocationID INT,
passenger_count INT,
trip_distance FLOAT,
total_amount INT,
taxi_type INT
);

INSERT OVERWRITE TABLE rides_lite_2018_2017
SELECT
YEAR(lpep_pickup_datetime) as year,
RatecodeID,
PULocationID,
DOLocationID,
passenger_count,
trip_distance,
total_amount,
CASE WHEN (REVERSE(SPLIT(REVERSE(INPUT__FILE__NAME), '[/]')[0])) LIKE
'green%' THEN 1 WHEN (REVERSE(SPLIT(REVERSE(INPUT__FILE__NAME),
'[/]')[0])) LIKE 'yellow%' THEN 2 END
FROM rides WHERE lpep_pickup_datetime LIKE '2018%' OR lpep_pickup_datetime
LIKE '2017%';
```

Construction and population of the table for the year 2018

```
CREATE TABLE rides_lite_2018 (
year INT,
RatecodeID INT,
PULocationID INT,
DOLocationID INT,
passenger_count INT,
trip_distance FLOAT,
total_amount INT,
taxi_type INT
);

INSERT OVERWRITE TABLE rides_lite_2018
```

```
SELECT
YEAR(lpep_pickup_datetime) as year,
RatecodeID,
PULocationID,
DOLocationID,
passenger_count,
trip_distance,
total_amount,
CASE WHEN (REVERSE(SPLIT(REVERSE(INPUT__FILE__NAME), '[/]')[0])) LIKE
'green%' THEN 1 WHEN (REVERSE(SPLIT(REVERSE(INPUT__FILE__NAME),
'[/]')[0])) LIKE 'yellow%' THEN 2 END
FROM rides WHERE lpep_pickup_datetime LIKE '2018%';
```

Query 4 for the table rides_lite_2018_2017

```
SELECT year, SUM(passenger_count) FROM rides_lite_2018_2017 GROUP BY year
HAVING year = '2018';
```

Query 5 for the table rides_lite_2018_2017

```
SELECT RatecodeID, SUM(total_amount)FROM rides_lite_2018_2017 WHERE year =
'2018' GROUP BY RatecodeID;
```

Query 6 for the table rides_lite_2018_2017

```
SELECT PULocationID, COLLECT_SET(DOLocationID) as group_doloc,
COLLECT_LIST(avg_trip) as group_trip FROM (SELECT PULocationID,
DOLocationID, CAST(FORMAT_NUMBER(AVG(trip_distance), 2) AS FLOAT) AS
avg_trip FROM rides_lite_2018_2017 WHERE taxi_type = 1 AND year = '2018'
GROUP BY PULocationID, DOLocationID ORDER BY avg_trip) table_temp GROUP BY
PULocationID;
```

Query 4 for the table rides_lite_2018

```
SELECT year, SUM(passenger_count) FROM rides_lite_2018 GROUP BY year;
```

Query 5 for the table rides_lite_2018

```
SELECT RatecodeID, SUM(total_amount)FROM rides_lite_2018 GROUP BY
RatecodeID;
```

Query 6 for the table rides_lite_2018

```
SELECT PULocationID, COLLECT_SET(DOLocationID) as group_doloc,
COLLECT_LIST(avg_trip) as group_trip FROM (SELECT PULocationID,
DOLocationID, CAST(FORMAT_NUMBER(AVG(trip_distance), 2) AS FLOAT) AS
avg_trip FROM rides_lite_2018 WHERE taxi_type = 1 GROUP BY PULocationID,
DOLocationID ORDER BY avg_trip) table_temp GROUP BY PULocationID;
```

Dealing with different operational modes

Another parameter to take in account when it comes to performance analysis is the use of the hive operational modes: local (exectution on the local machine as a simple node) and pseudo-distributed (simulation of a node in a network)

Activation of the local mode

```
SET mapred.job.tracker = local;
SET hive.exec.mode.local.auto = true;
```

Activation of the pseudo-distributed node

```
SET mapred.job.tracker = pseudo;
SET hive.exec.mode.local.auto = false;
```

Results

Query 4

Year	SUM(passenger_count)
2018	16789109123

Query 5

RatecodeID	SUM(total_amount)
3	135431705
35	393328
67	16894
99	4002
131	601
163	498
195	1433
547	0
611	44
18	38855698
50	87956
82	5783
114	2455
146	350
178	493
210	486
274	811
6098	10000
10	48361500
42	151954
74	10863
106	1746
138	466
20	16510505
52	78108

84	5190
116	1195
148	593
212	240
2	228543473
34	373812
66	28521
98	3858
130	1291
162	300
226	700
258	10005
290	729
6	49935919
38	257534
70	18206
102	1438
134	456
166	190
198	400
230	52
422	1059
25	1728785
57	42296
89	5796
121	931
153	1047
4	87835275
36	442198
68	18582
100	3300
132	1498
164	1
196	983

484	2417
932	14
1764	4414
12	24786499
44	135621
76	9504
108	685
140	763
204	1598
300	7
24	1983564
56	42497
88	3436
120	3028
152	277
216	302
248	412
600	5
5	70676335
37	344676
69	20014
101	10318
133	311
229	450
325	50
1061	8
2053	5134
5381	10001
22	5743867
54	59485
86	4615
118	3398
150	275
182	25

214	471
246	1322
31	584971
63	27367
95	3141
127	1363
159	782
831	11
7	41743771
39	225538
71	16419
103	3766
135	635
7655	10001
1	490008478
33	418931
65	27326
97	3820
129	2854
801	12
833	23
28	1648840
60	35305
92	4264
124	4166
156	66
188	10
252	1340
380	0
6204	10000
16	30579674
48	90516
80	4983
112	1801

144	510
176	2658
528	1325
4016	10000
19	23170653
51	75338
83	3399
115	1923
147	1051
211	1227
1459	3652
1811	4530
26	1874646
58	39006
90	4830
122	1150
154	462
218	338
250	656
602	18
1402	3509
1818	4551
17	49407742
49	90402
81	6218
113	2657
145	700
753	1887
1297	3247
1841	4606
0	206982960
32	539941
64	27540
96	5126

128	1353
27	1720835
59	36994
91	4318
123	1727
155	300
251	1177
603	26
1723	4311
14	15348514
46	101087
78	7740
110	2605
142	2405
206	581
270	25
302	234
910	300
23	2897899
55	56069
87	3805
119	2192
151	1822
183	0
2231	5581
9	52519319
41	172225
73	10075
105	1912
137	1021
169	372
329	0
13	16976114
45	119482

77	8688
109	3146
141	374
301	25
11	37619022
43	137584
75	6425
107	2154
139	1077
267	950
189483	4
29	1044709
61	33477
93	3641
125	1484
157	225
189	170
15	18439856
47	98777
79	8051
111	2853
207	502
239	639
943	14
21	11125701
53	61243
85	5157
117	3677
149	394
181	875
213	858
245	466
341	1002
8	47251101

40	188208
72	12932
104	2613
136	298
168	211
1928	4824
2408	6023
30	710537
62	30268
94	3368
126	1366
222	0
606	31
830	9

Query 6 (first row only)

PULocationID	COLLECT_SET(DOLocationID) AS group_doloc	COLLECT_LIST(avg_trip) AS group_trip
1	[1,265,23,249,66,68,14,164,22,243,145,166,257,64,132]	[1.06,8.93,12.7,12.86,15.95,17.1,18.71,20.1,22.0,23.9,25.21,26.64,32.9,34.9,37.1,161.19]

Performance comparison

The program instances are run on the following virtual machine:

The operative system is Ubuntu 18.4.4 hosted by VirtualBox vers. 6.0.14 which are given:

2 cores;
4096 MB RAM;
60GB disk space;

Equipped with Hadoop vers. 2.8.5, Java Open JDK vers. 1.8.0

The host machine with:

Intel© Core™ i7 6500U and SSD.

Time for completion

years: 2018
number of entries in table *rides_lite_2018*: 111609853

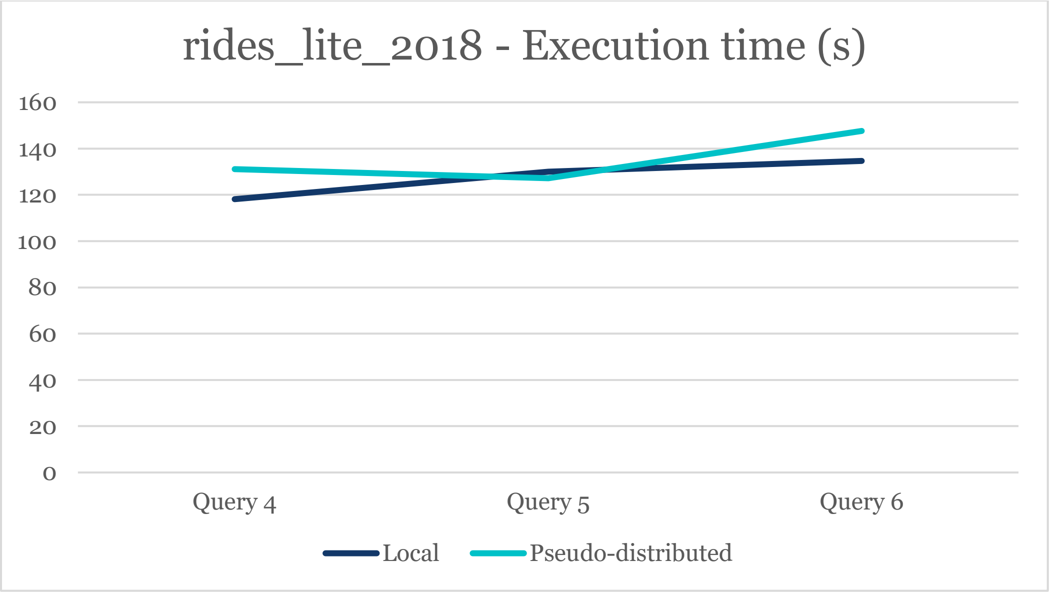
Task	Execution time in local mode (s)	Execution time in psuedo-distributed (s)
Query 4	118.137	131.078
Query 5	129.997	127.181
Query 6	134.667	147.617

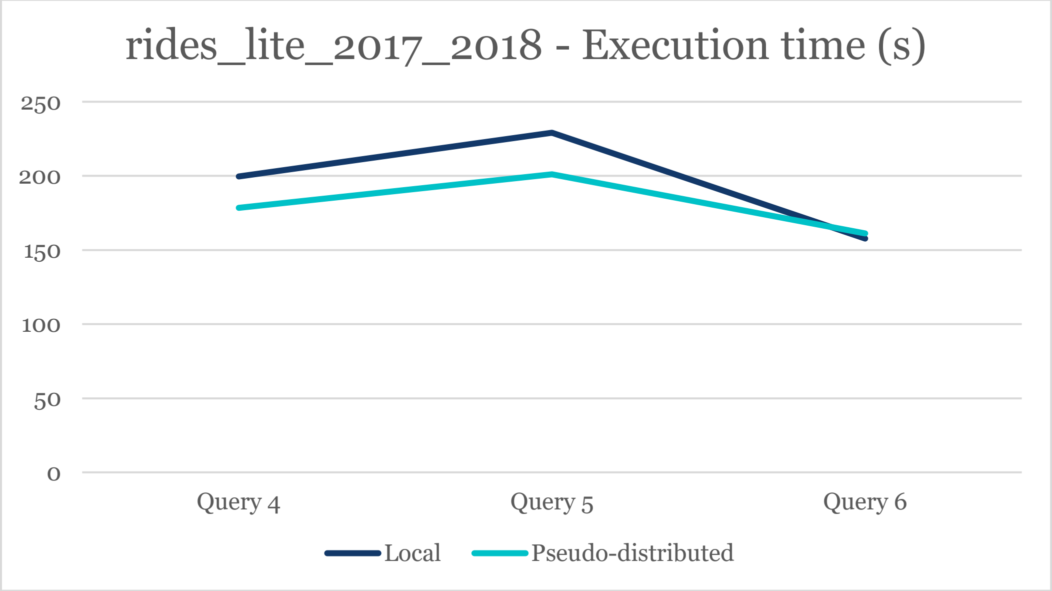
years: 2017, 2018
number of entries in table *rides_lite_2018_2017*: 236847300

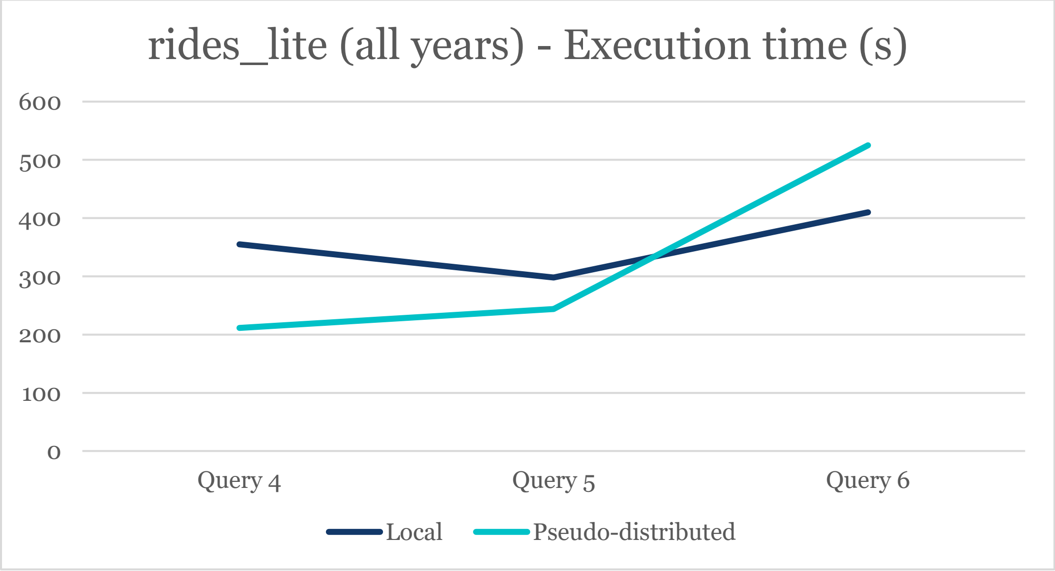
Task	Execution time in local mode (s)	Execution time in psuedo-distributed (s)
Query 4	199.514	178.525
Query 5	229.105	201.061
Query 6	157.725	161.184

years: 2016, 2017, 2018
number of entries in table *rides_lite*: 305975113

Task	Execution time in local mode (s)	Execution time in psuedo-distributed (s)
Query 4	354.845	211.382
Query 5	297.944	243.932
Query 6	409.915	525.028


rides_lite_2018 - Execution time (s)
160
140
120
100
80
60
40
20
0
Query 4
Query 5
Query 6
Local
Pseudo-distributed


rides_lite_2017_2018 - Execution time (s)
250
200
150
100
50
0
Query 4
Query 5
Query 6
Local
Pseudo-distributed


rides_lite (all years) - Execution time (s)
600
500
400
300
200
100
0
Query 4
Query 5
Query 6
Local
Pseudo-distributed

The pseudo-distributed mode performs better or worse than the local mode, depending on the type of query and the amount of data: the biggest differences are noticeable in case of the whole dataset table (made up of over 300M rows) where the pseudo-distributed mode takes -40.42 % time in comparison to local mode in terms of completion time for the query 4, while it uses +28.08 % time more when the query 6 is run.

Limitations

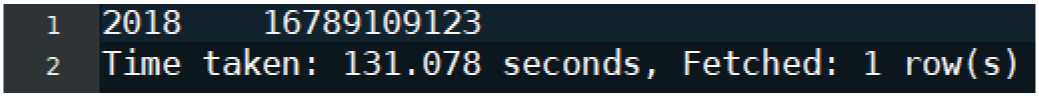
The results concerning the execution times are biased by the state of the host machine which was loaded with different tasks in background, causing a not constant access time and bandwidth availability on the SSD on the host operating system, so these must be taken as an overall idea of the average execution time for the machine used.

References

[1] https://en.wikipedia.org/wiki/New_York_City_Taxi_and_Limousine_Commission
[2] https://www.quora.com/What-is-the-difference-between-Green-Cabs-and-Yellow-Cabs
[3] https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf
[4] https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_green.pdf
[5] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+VirtualColumns

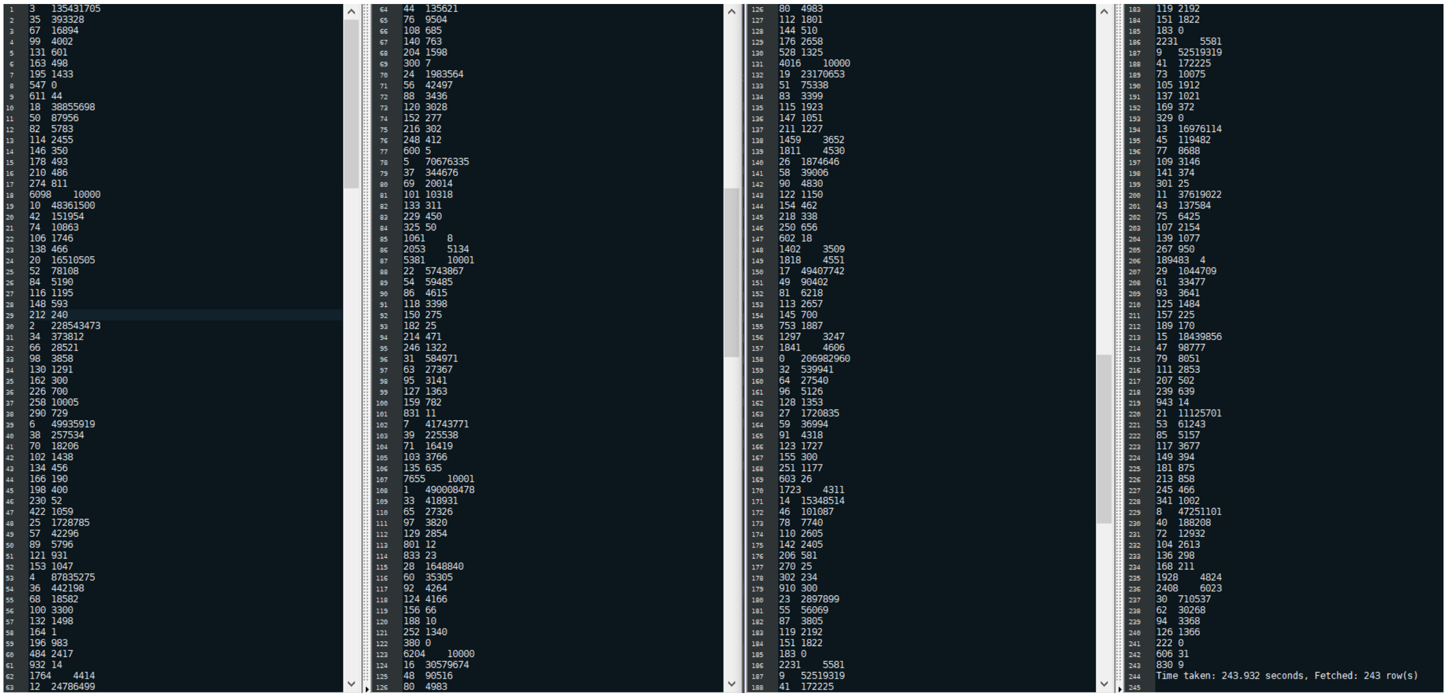
Appendix

Screenshot of resulting file for Query 4



```
2018    16789109123
Time taken: 131.078 seconds, Fetched: 1 row(s)
```

Screenshot of resulting file for Query 5



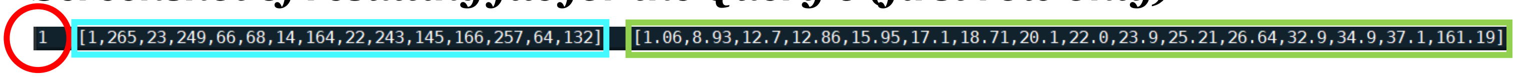
```
3   135431705
35  393328
67  16894
99  4002
131 601
163 498
195 1433
547 0
611 44
18  38855698
50  87956
82  5783
114 2455
146 350
178 493
210 486
274 811
6098    10000
10  48361500
42  151954
74  10863
106 1746
138 466
20  16510505
52  78108
84  5190
116 1195
148 593
212 240
2   228543473
34  373812
66  28521
98  3858
130 1291
162 300
226 700
258 10005
290 729
6   49935919
38  257534
70  18206
102 1438
134 456
166 190
198 400
230 52
422 1059
25  1728785
57  42296
89  5796
121 931
153 1047
4   87835275
36  442198
68  18582
100 3300
132 1498
164 1
196 983
484 2417
932 14
1764    4414
12  24786499
```

```
44  135621
76  9504
108 685
140 763
204 1598
300 7
24  1983564
56  42497
88  3436
120 3028
152 277
216 302
248 412
600 5
5   70676335
37  344676
69  20014
101 10318
133 311
229 450
325 50
1061    8
2053    5134
5381    10001
22  5743867
54  59485
86  4615
118 3398
150 275
182 25
214 471
246 1322
31  584971
63  27367
95  3141
127 1363
159 782
831 11
7   41743771
39  225538
71  16419
103 3766
135 635
7655    10001
1   490008478
33  418931
65  27326
97  3820
129 2854
801 12
833 23
28  1648840
60  35305
92  4264
124 4166
156 66
188 10
252 1340
380 0
6204    10000
16  30579674
48  90516
80  4983
```

```
80  4983
112 1801
144 510
176 2658
528 1325
4016    10000
19  23170653
51  75338
83  3399
115 1923
147 1051
211 1227
1459    3652
1811    4530
26  1874646
58  39006
90  4830
122 1150
154 462
218 338
250 656
602 18
1402    3509
1818    4551
17  49407742
49  90402
81  6218
113 2657
145 700
753 1887
1297    3247
1841    4606
0   206982960
32  539941
64  27540
96  5126
128 1353
27  1720835
59  36994
91  4318
123 1727
155 300
251 1177
603 26
1723    4311
14  15348514
46  101087
78  7740
110 2605
142 2405
206 581
270 25
302 234
910 300
23  2897899
55  56069
87  3805
119 2192
151 1822
183 0
2231    5581
9   52519319
41  172225
```

```
119 2192
151 1822
183 0
2231    5581
9   52519319
41  172225
73  10075
105 1912
137 1021
169 372
329 0
13  16976114
45  119482
77  8688
109 3146
141 374
301 25
11  37619022
43  137584
75  6425
107 2154
139 1077
267 950
189483  4
29  1044709
61  33477
93  3641
125 1484
157 225
189 170
15  18439856
47  98777
79  8051
111 2853
207 502
239 639
943 14
21  11125701
53  61243
85  5157
117 3677
149 394
181 875
213 858
245 466
341 1002
8   47251101
40  188208
72  12932
104 2613
136 298
168 211
1928    4824
2408    6023
30  710537
62  30268
94  3368
126 1366
222 0
606 31
830 9
Time taken: 243.932 seconds, Fetched: 243 row(s)
```

Screenshot of resulting file for the Query 6 (first row only)



```
1   [1,265,23,249,66,68,14,164,22,243,145,166,257,64,132]   [1.06,8.93,12.7,12.86,15.95,17.1,18.71,20.1,22.0,23.9,25.21,26.64,32.9,34.9,37.1,161.19]
```

Row composition for Query 6 file

Red: PULocationID
Light blue: list of DOLocationID
Green: sorted list of trip_distance in ascending order relative to DOLocationID