CS-236 SIMBA Project Report

Qihua Liang - 861195605 Lecturer: Prof. Vassillis Tsotras
Gautham Mani - 862002022

1. Introduction

In this project, we explored Spark and Simba. We ran a series of tasks and queries according to the project requirements. During this process, we encountered a bunch of problems and solved them. Some findings are also quite interesting.

2. Before Running the Code

We found that the vividsolutions library package has some naming issue in the Simba source code that we checked out from Github. So we had to manually download the com.vividsolutions.jts-1.13 jar and stored it in the Spark jar folder. You might need to do the same thing before running the code.

Also, you need to put the trajectories.csv and POI.csv datasets in the spark folder.

3. How to Run the Code

First, unzip the project folder. Then go into the project folder. Run sbt package
Second, go into the spark folder
Then

- For part1, run bin/spark-submit –class org.apache.spark.sql.simba.examples.RTreeIndex [path of the compiled source code jar]

Then you need to run ./plot.py [sparkpath/mbrs.csv] [sparkpath/points.csv]

- For part2 query one, run bin/spark-submit -class org.apache.spark.sql.simba.examples.queryOne [path of the compiled source code jar].
- For part2 query two, run bin/spark-submit -class org.apache.spark.sql.simba.examples.queryTwo [path of the compiled source code jar].
- For part2 query three, run bin/spark-submit -class org.apache.spark.sql.simba.examples.queryThree [path of the compiled source code jar].
- For part2 query four, run bin/spark-submit -class org.apache.spark.sql.simba.examples.queryFour [path of the compiled source code jar].
- For part2 query five, run bin/spark-submit -class org.apache.spark.sql.simba.examples.queryFive [path of the compiled source code jar].

- For part3, we tried to write a wrapper on top of query 4 and query 5, but ended up with memory size issues. So we had to manually run all parameters. Steps are as follow:

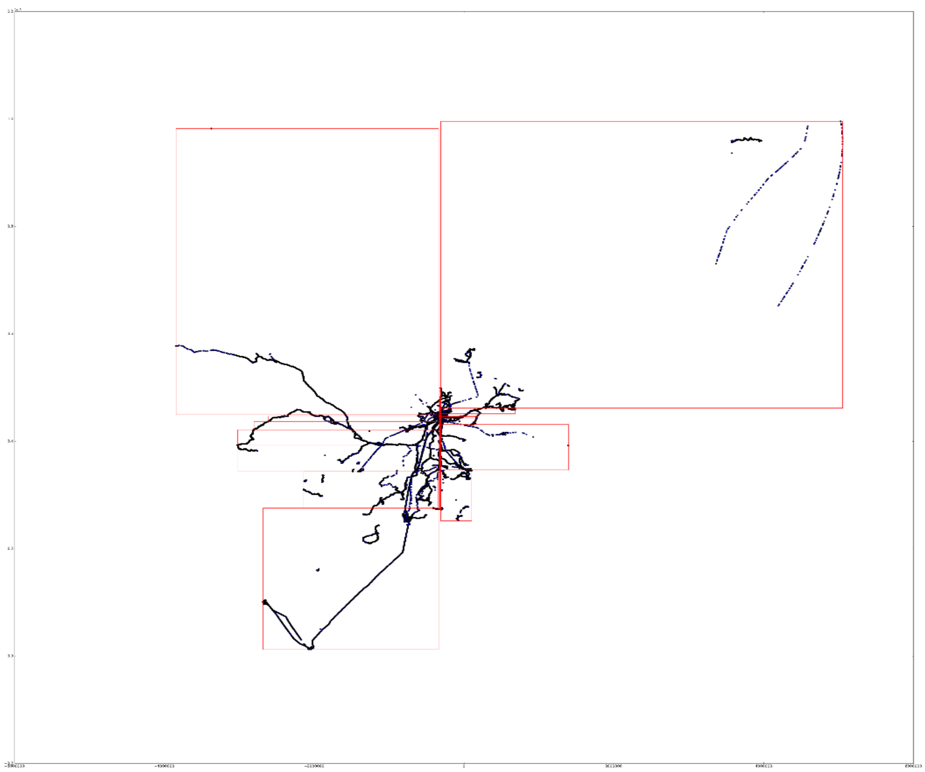
- Change a combination of parameters. E.g. 100m and 4 cores
- Recompile by sbt package
- bin/spark-submit -class org.apache.spark.sql.simba.examples.queryFour
- bin/spark-submit -class org.apache.spark.sql.simba.examples.queryFive

4.1. Part One

Our approach of implementing this par is as follows:

1. Read 10% of the data into data frame by using spark sample() method with replacement
2. Use Simba method to build the RTreeIndex
3. Loop through all MBRs using the mapPartitions() method and write all mbrs into mbrs.csv in the format of (small longitude, large longitude, small latitude, large latitude)
4. Loop through sampled data frame and write all points in the format of (longitude, latitude) into points.csv
5. Use python matplotlib library to plot the result

The plot is as following:



We also submitted this file in the result/plot.png

4.2.1. Query One

Our approach of implementing this is as following:

1. Read data frame
2. Run a Simba range query to filter out points that are out of the 5^{th} ring region
3. Then filter the data frame by checking if it contains “amenity=restaurant” in the description
4. Write result to the queryOne.csv file

We submitted this file in the result/queryOne.csv

4.2.2. Query Two

Our approach of implementing this is as follows:

1. Read data frame and transform the string format timestamp into timestamp type
2. Add two new columns to store the “day of week” and hour using date_format(timestamp, “EEEE”) and hour(timestamp)
3. Run Simba circleRange query to get all points within 2km
4. Use spark groupBy(hour) on the data, and then aggregate it by using the countDistinct(trajectory id, object id)
5. The result is as following. We also write result to queryTwo.csv

```
+----+-----+
|hour|count|
+----+-----+
|   0|   88|
|   1|   69|
|   2|   92|
|   3|   78|
|   4|   82|
|   5|   83|
|   6|   92|
|   7|   94|
|   8|  103|
|   9|  116|
|  10|  199|
|  11|  132|
|  12|  103|
|  13|   60|
|  14|   56|
|  15|   12|
|  16|   18|
|  17|    4|
|  18|   10|
|  21|   11|
|  22|    7|
|  23|   21|
+----+-----+
```

We also submitted this file in the result/queryTwo.csv

4.2.3. Query Three

Our approach of implementing this is as follows:

1. Read data frame
2. Do a Simba range query to filter out points that are out of the 5^{th} ring region
3. Add a new column to map a point to a quadrant with index 0 to 3
4. Run an aggregation to the data frame to find out the earliest and latest timestamp for all trajectories
5. Join 4 and 3 to get the intermediate table as following

id	min	max	startQuadrant	endQuadrant
311	2009-03-17 08:25:...	2009-03-17 08:35:...	1	1
423	2009-04-04 08:05:...	2009-04-04 08:15:...	1	1
451	2009-03-02 04:13:...	2009-03-02 04:18:...	1	1
695	2009-03-23 10:32:...	2009-03-23 10:35:...	1	1
850	2009-03-09 11:36:...	2009-03-09 11:41:...	3	3
862	2009-03-16 03:00:...	2009-03-16 03:04:...	1	1
1046	2009-02-15 08:49:...	2009-02-15 08:49:...	3	3
1152	2009-07-10 10:46:...	2009-07-10 10:50:...	0	0
1595	2009-02-18 06:51:...	2009-02-18 06:56:...	1	1
1848	2009-07-19 22:55:...	2009-07-19 22:56:...	1	1
2018	2009-02-27 12:57:...	2009-02-27 12:58:...	3	3
3715	2011-06-23 23:46:...	2011-06-23 23:47:...	1	1
4315	2011-06-21 11:25:...	2011-06-21 11:27:...	3	3
4647	2011-08-11 11:42:...	2011-08-11 11:43:...	3	3
6386	2011-08-14 23:18:...	2011-08-14 23:19:...	1	1
6513	2011-07-05 11:00:...	2011-07-05 11:02:...	1	1
6543	2011-08-17 23:04:...	2011-08-17 23:06:...	3	3
6569	2011-06-28 23:02:...	2011-06-28 23:04:...	3	3
6891	2011-08-08 11:12:...	2011-08-08 11:13:...	3	3
6909	2011-07-16 00:40:...	2011-07-16 00:41:...	3	3

only showing top 20 rows

6. Add a new column sameQuadrant which means where the start and end points are in the same quadrant. When startQuadrant == endQuadrant, it should be set to true, otherwise we should get false
7. Run aggregation to get the count. Final result is as follow

sameQuadrant	count
true	155530
false	7321

4.2.4. Query Four

Our approach for implementing this is as follows:

1. Read the Trajectories.csv into a data frame and transform the string format for TID and OID to Long type, LON and LAT to Double type and Time into timestamp type
2. Make a copy of the Trajectories.csv into another data frame and transform the string format to the data types mentioned in (1) for each attribute
3. For the second copy rename the LON and LAT column so that it is not the same as the LON and LAT columns in the first copy of the dataset
4. Take a sample for both the data sets
5. Index the datasets in using R-Tree based on LON and LAT field of both the datasets
6. Create a new column which extracts only the date from the time column for both the data sets
7. Perform FILTER operation on both the datasets using the new date column for the months of FEBRUARY to JUNE for all years present in the datasets
8. Perform a DISTANCEJOIN using the filtered datasets based on the LON and LAT columns of both the datasets using a radius of 100 meters

9. Perform DISTINCT operation on the resulting dataset to remove duplicate data points
10. Perform a GROUPBY operation on the LON and LAT of the resulting dataset of the previous operation
11. Perform COUNT and ORDERBY operations to get the points with more number of points around them within a radius of 100 meters.

Due to the large amount of data and the memory restrictions on my laptop I have run the query for each year and the results

t_lon	t_lat	count
-326661.288813	4471892.82933	18472
-326646.616105	4471908.11915	18460
-326641.11025	4471905.63913	18433
-326646.413037	4471909.97426	18403
-326675.16071	4471911.24383	18239
-326658.428595	4471879.37492	18055
-326686.375476	4471914.34883	17863
-326690.657167	4471914.81755	17654
-326679.85425	4471921.14429	17622
-326664.972744	4471925.14723	17584
-326690.454088	4471916.67265	17535
-326670.681654	4471925.77217	17529
-326690.251009	4471918.52776	17425
-326693.308548	4471916.98513	17357
-326702.887335	4471908.64707	17312
-326701.042601	4471885.91712	17252
-326712.059976	4471904.01922	17152
-326680.46917	4471928.72093	17130
-326676.164647	4471875.68442	17110
-326607.668952	4471894.46929	17048

Top 20 points for year 2007

t_lon	t_lat	count
-316556.838686	4471349.85886	73656
-324701.477161	4472780.51467	56230
-326643.991959	4471912.29995	55086
-324684.731123	4472814.84048	46680
-324687.111243	4472807.32882	46536
-324690.0233	4472790.18884	45968
-326906.182712	4472583.73483	40608
-316170.260682	4472277.55339	38592
-326660.522512	4471893.49638	37568
-326680.952984	4471908.49495	36974
-326641.547965	4471906.99742	36956
-326665.051927	4471917.8268	36668
-326667.172436	4471882.62241	36616
-326683.887954	4471892.29196	36504
-326662.350638	4471922.79138	36034
-326676.759768	4471923.05083	35884
-326656.500556	4471924.7755	35648
-326681.679138	4471924.34404	35576
-326697.308073	4471910.84854	35412
-326657.520662	4471927.33145	35334

Top 20 points for year 2008

t_lon	t_lat	count
-326436.634523	4471678.51925	27579
-326437.5887	4471678.51102	27564
-326434.445471	4471705.65095	18590
-326443.364144	4471707.30275	18564
-326442.28443	4471700.5389	18548
-326433.346066	4471683.22814	18430
-326440.45273	4471681.6404	18420
-326436.451071	4471678.61181	18388
-326442.445595	4471679.26779	18382
-326434.946537	4471678.89773	18380
-326420.512845	4471681.71119	18338
-326435.874476	4471675.16955	18330
-326435.544112	4471675.02076	18330
-326427.300192	4471677.49781	18330
-326449.651205	4471679.94366	18326
-326437.366455	4471674.20643	18322
-326438.418449	4471674.09627	18318
-326436.473178	4471673.65812	18314
-326433.768987	4471672.23582	18288
-326440.730249	4471673.56077	18284

Top 20 points for year 2008

t_lon	t_lat	count
-326676.17061	4471887.50475	35852
-326691.516794	4471901.76657	35366
-326660.525801	4471929.09094	34658
-326716.214371	4471917.60409	32676
-326673.026084	4471863.50615	31268
-326596.992967	4471910.98514	30170
-326470.383926	4471897.83746	22904
-326903.948682	4471933.1009	22564
-326817.162309	4471917.62844	21428
-326952.719656	4471898.45388	18554
-326952.634022	4471898.44451	18548
-326658.77707	4471902.3165	18470
-326648.182144	4471899.35449	18437
-326671.291684	4471900.48746	18406
-326652.93534	4471892.66585	18397
-326660.790918	4471897.45689	18387
-326663.943314	4471900.24626	18383
-326666.600812	4471895.84008	18351
-326673.47315	4471906.92146	18336
-326657.218304	4471890.43132	18335

Top 20 points for year 2010

t_lon	t_lat	count
-326480.290016	4472267.819	68180
-326481.818699	4472267.04767	48048
-326476.062478	4472251.02357	41538
-326679.619467	4471896.901	36856
-326677.537825	4471892.16753	36788
-326683.064564	4471895.77626	36522
-326682.84004	4471895.18848	36512
-326482.06352	4472267.44993	34305
-326481.166616	4472267.72722	34210
-326471.746215	4472264.06786	33270
-326468.645912	4472260.72481	33055
-326522.942283	4472273.9895	32128
-326461.175445	4472286.75285	30560
-326990.787217	4471957.06577	28494
-326476.368215	4472250.8693	27732
-326476.245804	4472250.66817	27724
-326486.897287	4472272.10912	27700
-326475.800262	4472257.37777	27460
-326480.453037	4472267.64912	27316
-326480.7814	4472272.5661	27052

Top 20 points for year 2011

4.2.5. Query Five

Our approach for implementing this is as follows

1. Read the Trajectories.csv into data frame and transform the string format for ID and OID to Long type, LON and LAT to Double type and TIME into timestamp type for Trajectories.
2. Read POIs.csv into data frame and transform string format for PID to Long type, LON and LAT to Double type
3. Generate new columns for DAY, DATE, MONTH, YEAR and HOUR from the TIME column of the Trajectories data set
4. Separately Index Trajectories and POIs dataset using R-Tree with LON and LAT columns of each dataset
5. Perform DISTANCEJOIN operation using the LON and LAT columns of the two datasets with a radius of 100 meters
6. Perform FILTER on the joined dataset using the DAY column to get records with only workdays
7. Perform two FILTER operations, for year 2008 and 2009, separately on the previous results using the YEAR column to get records which only occur during the two years
8. Perform a GROUPBY operation for each year on LON, LAT and MONTH columns followed by a COUNT operation to get the count of popular points
9. Perform an ORDERBY operation on the results of each year to get the top 10 popular places for the respective years with the highest number of objects within a radius of 100 meters.

Due to the large amount of data and the memory restrictions on my laptop I have run the query on a sample of the trajectories.csv

poi_lon	poi_lat	month	count
-326771.872576809	4472606.65939349	11	2476
-326792.860175067	4472636.09284529	11	2058
-322216.795602612	4474247.65871495	07	1611
-326815.825293712	4472636.15219576	11	1534
-327185.304985578	4475582.42470776	12	1273
-322305.400843433	4474240.61192863	07	1237
-329107.780598999	4473533.41600478	08	1128
-326408.161626098	4471834.49459609	12	1083
-326700.973247766	4466430.85087653	11	1039
-325586.899594995	4470830.95939808	10	1019

Top-10 Results for year 2008

poi_lon	poi_lat	month	count
-326501.387617249	4473630.543939	03	2085
-326491.880994424	4473603.89979181	03	2027
-326490.153359661	4473585.65438308	03	1902
-326486.271813817	4473573.46970909	03	1570
-326500.869801015	4473685.19662871	03	1533
-326501.387617249	4473630.543939	04	1497
-329238.569658378	4472531.67593712	03	1443
-326485.727442932	4473566.65168839	03	1410
-326491.880994424	4473603.89979181	04	1400
-326487.45405735	4473559.42903025	03	1381

Top-10 Results for year 2009

4.3. Part Three

For part three of the project we have used the Programs from Query 4 and Query5. We have performed the experiments for different number of cores as well as by varying the radii

4.3.1. Query Four

Time Taken(in milliseconds)				
No of Cores	1	2	3	4
Radius(meters)				
100	779052	407985	372380	356035
200	829078	406543	361617	353662
300	820759	465969	414345	401918
400	831664	500511	478324	435906
500	949005	567332	395984	373968

4.3.2. Query Five

Time Taken(in milliseconds)				
No of Cores	1	2	3	4
Radius(meters)				
100	118773	70979	69153	59585
200	116781	69515	65460	60974
300	115497	72817	69796	59982
400	119202	71374	65021	59149
500	117735	68871	66341	58633

5. Conclusion

From the above table we can see that:

- As the number of cores increases the time taken to process the query decreases
- As the radius increases the time taken for processing the query increases

Based on the above two observations, we can conclude that the time taken to complete each query decreases as the number of cores increases for each corresponding radius chosen.

6. Acknowledgement

We would like to thank Andres Calderon for being available at all times and clear our doubts with SIMBA and the project as a whole. Finally, we would like to thank Prof. Vassilis Tsotras for giving us an opportunity to learn and get a hands on experience in using a new database analytics tool and imparting knowledge about various concepts and techniques in DataBase Management Systems.

7. Reference

1. Spatial In-Memory Big-Data Analytics - http://www.cs.utah.edu/~dongx/simba/
2. Apache Spark Documentation - https://spark.apache.org/
3. Apache Spark Scala Documentation - https://spark.apache.org/docs/0.9.1/scala-programming-guide.html/
4. SIMBA Source Code - https://github.com/InitialDLab/Simba
5. Miscellaneous Queries - https://stackoverflow.com/