CSE 511: Group 18
System Documentation

Aravamuthan Lakshminarayanan
1211111111

Muffakham Ali Farhan Mohammed
1211111111

Rahul Manghnani
1211111111

Shahabudeen Sajith
1216259811

Vivek Tiwary
1211111111

1 December 2019

Contents

1 Tasks

1.1 Phase 1

The tasks for Phase 1 involved writing user-defined functions in SparkSQL and using them to perform spatial queries.

The **functions** to be implemented are as follows:

1.1.1 `ST_Contains(queryRectangle, pointString)`

`ST_Contains(queryRectangle: String, pointString: String): Boolean`

Parse the `pointString` and `queryRectangle` and check whether the `queryRectangle` fully contains the point.

1.1.2 `ST_Within(pointString1, pointString2, distance)`

Parse the `pointString1` and `pointString2` and check the two points are within the given (Euclidean) `distance`.

The **spatial queries** to be performed are as follows:

1.1.3 `Range`

Given a query rectangle `R` and a set of points `P`, find all the points within `R`.

1.1.4 `Range Join`

Given a set of Rectangles `R` and a set of Points `S`, find all `(Point, Rectangle)` pairs such that the point is within the rectangle.

1.1.5 `Distance`

Given a point location `P` and distance `D`, find all points that lie within a distance `D` from `P`.

1.1.6 `Distance Join`

Given a set of Points `S1` and a set of Points `S2` and a distance `D`, find all `(s1, s2)` pairs such that `s1` is within a distance `D` from `s2`.

1.2 Phase 2

The tasks for Phase 2 involved performing spatial hot spot analysis.

1.2.1 Hot zone analysis

The hotness of a rectangle is defined as the count of the number of points located inside it. Given a rectangle and points dataset, the task is to calculate the hotness of all the rectangles.

1.2.2 Hot cell analysis

This task was to apply spatial statistics to spatio-temporal big data in order to identify statistically significant spatial hot spots. Concretely, compute a list of the fifty most significant hot spot cells in time and space as identified using the Getis-Ord G_i^* statistic

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j}x_j - \bar{X}\sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{[n\sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2]}{n-1}}} \tag{1}$$

where x_j is the attribute value for cell j , $w_{i,j}$ is the spatial weight between cell i and j, n is equal to the total number of cells, and:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n} \tag{2}$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2} \tag{3}$$

The G_i^* statistic is a z-score. The neighborhood for each cell in the space-time cube is established by the neighbors in a grid based on subdividing latitude and longitude uniformly. This spatial neighborhood is created for the preceding, current, and following time periods (i.e., each cell has 26 neighbors). For simplicity of computation, the weight of each neighbor cell is presumed to be equal.

2 Environment

The code is written in Scala to run against Apache Spark. IntelliJ is the IDE of choice. A system with at least 8 GB of RAM and 128 GB of Storage is recommended for locally running the Spark application.

2.1 Requirements

1. JDK 1.8.0
2. Python 3.6
3. Scala 2.11.12
4. Spark 2.4.4
5. sbt 0.13.17

2.2 Path

```
export JAVA_HOME=~/jdk1.8.0_162.jdk/Contents/Home
export SPARK_HOME=~/spark-2.4.4-bin-hadoop2.7
export SBT_HOME=~/sbt
export SCALA_HOME=~/scala-2.11.12
export PYSPARK_PYTHON=python3
export PATH="~/Python.framework/Versions/3.6/bin":$PATH
export PATH=$JAVA_HOME/bin:$PATH
export PATH=$SBT_HOME/bin:$PATH
export PATH=$SBT_HOME/lib:$PATH
export PATH=$SCALA_HOME/bin:$PATH
export PATH=$SCALA_HOME/lib:$PATH
export PATH=$SPARK_HOME/bin:$PATH
export PATH=$SPARK_HOME/lib:$PATH
```

2.3 Source Code

The source code for the entire project is hosted as a private repository over Github. https://github.com/hexhog/CSE511_Project

2.4 IntelliJ Setup

1. Import the project into the IDE.
2. Select the JDK and Scala version when prompted.
3. Once the project is up and the dependencies have been installed, right click on the main class and select run. The main class for Phase 1 is `SparkSqlExample.scala` and the main class for Phase 2 is `Entrance.scala`
4. Append `.master("local[]")` after `.config("spark.some.config.option", "some-value")` to tell IDE the master IP is localhost.

2.5 Submitting the code to Spark

1. Go to project root folder.
2. Run `sbt clean assembly`
3. Find the packaged jar in `target/scala-2.11/`
4. Submit the jar to Spark using Spark command `spark-submit target/scala-2.11/_.jar`.

2.6 Testing on a cluster locally

1. Start the master `spark-2.4.4-bin-hadoop2.7/sbin/start-master.sh`
2. Spawn workers in separate shells `spark-2.4.4-bin-hadoop2.7/bin/spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077 -c 1 -m 1024M`
3. Submit the jar to the master using `spark-submit --master spark://localhost:7077 target/scala-2.11/_.jar`.

3 Phase 1

3.1 Implementation

3.1.1 ST_Contains(queryRectangle, pointString)

ST_Contains first computes the four corner points, each as a pair of coordinates, for the given queryRectangle. The function then checks if the coordnate pair given by pointString lies in between these coordinates.

3.1.2 ST_Within(pointString1, pointString2, distance)

ST_Contains computes the Euclidean distance between the two coordinates given by pointString1 and pointString2. This distance is then compared with distance.

3.1.3 Range

Range calls ST_Contains on the given set of points and rectangle.

3.1.4 Range Join

Range Join calls ST_Contains on the given set of points and the set of rectangles.

3.1.5 Distance

Distance calls ST_Within on the source point and the given set of points.

3.1.6 Distance Join

Distance Join calls ST_Within between the two sets of points.

3.2 Testing

The code was tested against the given sample example input.

```
spark-submit target/scala-2.11/Phase1.jar result/output
rangequery src/resources/arealm10000.csv
-93.63173,33.0183,-93.359203,33.219456
rangejoinquery src/resources/arealm10000.csv
src/resources/zcta10000.csv
distancequery src/resources/arealm10000.csv
-88.331492,32.324142 1
distancejoinquery
src/resources/arealm10000.csv src/resources/arealm10000.csv
0.
```

The output matched the given sample output.

```
4 7612 302 123362
```

4 Phase 2

4.1 Implementation

4.1.1 Hot zone

To be added

4.1.2 Hot cell

To be added

4.2 Testing

The code was tested against the given sample example input.

```
spark-submit target/scala-2.11/Phase2.jar result/output
hotzoneanalysis src/resources/point-hotzone.csv
src/resources/zone-hotzone.csv
hotcellanalysis src/resources/yellow_tripdata_2009-01_point.csv
```

The output matched the given sample output.

5 Task Distribution

5.1 Aravamuthan Lakshminarayanan

1.

5.2 Muffakham Ali Farhan Mohammed

1.

5.3 Rahul Manghnani

1.

5.4 Shahabudeen Sajith

1. Shared documentation to setup the environment.
2. Implemented Phase 1 of the project.
3. Tested Phase 2 of the project on local cluster setup.
4. Contributed to System Documentation report.

5.5 Vivek Tiwary

1.