CSE 512 Assignment 2

Maximum points Possible – 10

The required task is to understand and implement two tasks: i) Parallel spatial joining technique on-top of an open-source relational database management system (i.e., PostgreSQL) using PostGIS spatial extension and ii) Spatial joining with a map reduce program on top of Apache Spark using the Apache Sedona spatial extension.

Input Data: Input datasets consist of two spatial datasets: a point dataset and a rectangle dataset. The point dataset (points.csv) consists of latitudes and longitudes of various points while the rectangle dataset (rectangles.csv) consists of latitudes and longitudes of two diagonal points of rectangles.

Each row in points.csv file has the format longitude,latitude while the same for the rectangles.csv file is longitude1,latitude1,longitude2,latitude2.

Our goal is to perform spatial join between points dataset and rectangles dataset and return the number of points inside each rectangle (including points on rectangle boundary).

Required Tasks

Part A (Parallel Spatial Join with PostGIS)

We load the points and rectangles datasets into two PostgreSQL tables: *points* and *rectangles* respectively. The *points* table has a *geometry* type column *geom* which denotes *Point* geometry. The *rectangles* table has a *geometry* type column *geom* which denotes *Envelop* geometry (rectangle). You are given a Python file *Assignment2_Interface.py* with an incomplete function *parallelJoin*. You need to complete this function. It has following attributes: *pointsTable*, *rectsTable*, *outputTable*, *outputPath*, and *openConnection*. The high-level objective is to find the number of points inside each rectangle save it to the *outputTable* and *outputPath*. Perform the following tasks:

- Create four partitions/fragments of both *pointsTable* and *rectsTable*. You should consider space partitioning such that all points or rectangles of a partition should lie within the corresponding fragment. Fragments doesn't need to satisfy disjointness property.
- Run four parallel threads. Each thread should perform a spatial join between a fragment of *pointsTable* and a fragment of *rectsTable*. The purpose of the join is to find the number of points (*pointsTable.geom*) inside each rectangle or Envelop (*rectsTable.geom*) within the corresponding fragment. You must make use of ST_Contains method supported by PostGIS.
- Sort the output of each fragment in the ascending order of counts of points inside the parallel threads.
- Merge the outputs of four parallel joins into *outputTable* in the ascending order of counts of points. You can design the structure of the *outputTable* as you wish, but it should have a column named points_*count* containing counts of points.

- Write the counts of points into the *outputPath* in the ascending order. You don't need to write rectangle coordinates.

Set up Instructions for Part A: You should have PostgreSQL >=13 installed on your device. On top of it, you should install PostGIS extension.

Naming Conventions to be Followed: You should not change the following naming conventions:

- Database Name: dds_assignment2
- Postgres Username: postgres
- Postgres Password: 12345

Part B (Spatial Join with Map Reduce on Apache Sedona)

Similar to Part A, you need to perform a spatial join between the points dataset and rectangles dataset in order to count the number of points located within each rectangle. This time, you will perform the task with the help of map and reduce functionalities supported by Spark.

Task: You are given a Apache Sedona project named Map-Reduce-Apache-Sedona. Find the Scala file *SparkMapReduce.scala*. Complete the incomplete function *runMapReduce*(). Instead of using group by operation, you must make use of Spark *map* and *reducebyKey* operations to complete the task. Covert the resultant RDD back to a DataFrame before returning the result. The returned DataFrame should contain the number of points within each rectangle in an ascending order of count values. You don't need to return rectangles.

Setting Up Hadoop and Apache Spark Test Environment:

The following setup instructions are specific to Ubuntu operating system:

- Install Java version >= 1.8 and set up the JAVA_HOME environment variable. If you don't know how to set up the JAVA_HOME environment variable, follow the link: https://askubuntu.com/questions/175514/how-to-set-java-home-for-java. To check whether set up is done, run the command *echo $JAVA_HOME* on your command line. You should see the path.
- Download Hadoop-2.7.7 from the link: https://archive.apache.org/dist/hadoop/common/hadoop-2.7.7/hadoop-2.7.7.tar.gz and extract it to your desired location. Now, you need to set up the HADOOP_HOME environment variable. Run the command *sudo nano ~/.bashrc* on the command line. Copy the statement *export HADOOP_HOME=path to the folder hadoop-2.7.7* in the opened file. Save and close the file. Run the command *source ~/.bashrc* and check the correctness of the setup with *echo $HADOOP_HOME* command.

- Download Spark-2.4.7 from the link: https://drive.google.com/file/d/1XlRcJNoGTaf-0YvVv8pwZzzjrCqG-IRA/view?usp=sharing and extract it to your desired location. Now, you need to set up the SPARK_HOME environment variable. Run the command *sudo nano ~/.bashrc* on the command line. Copy the statement *export SPARK_HOME=path to the folder spark-2.4.7-bin-hadoop2.7* in the opened file. Save and close the file. Run the command *source ~/.bashrc* and check the correctness of the setup with *echo $SPARK_HOME* command. If you browse the path to the SPARK_HOME, you should see *spark-submit* under the *bin* folder which is required for your testing.

How to submit your code to Spark:

- Go to project root folder
- Run *sbt clean assembly*. You may need to install sbt in order to run this command.
- Find the packaged jar in "./target/scala-2.11/Map-Reduce-Apache-Sedona-assembly-0.1.jar"
- Now, you can run the jar on Spark with spark-submit command. If you already have set up the Hadoop and Spark test environment, you should be able to run the spark-submit command from your command line. Submit the jar to Spark using Spark command "./bin/spark-submit". A pseudo code example: ./bin/spark-submit PATH_TO_JAR_Map-Reduce-Apache-Sedona-assembly-0.1.jar output mapreduce PATH_TO_points.csv PATH_TO_rectangles.csv

How to debug your code in IDE:

- Use IntelliJ Idea with Scala plug-in or any other Scala IDE.
- In some cases, you may need to go to "build.sbt" file and change % "provided" to % "compile" in order to debug your code in IDE
- Run your code in IDE
- **You must revert Step 2 and 3 above and recompile your code before use spark-submit!!!**

Submission Instructions:

- You should submit a .zip file containing two sub folders: Part-A and Part-B. Folder Part-A should only have the file *Assignment2-Interface.py* Folder Part-B contains 1) jar file *Map-Reduce-Apache-Sedona-assembly-0.1.jar* and 2) Apache Sedona project *Map-Reduce-Apache-Sedona*. Don’t put the datasets inside your zip file. Once you compile the Apache Sedona project, two target folders are generated. One inside the root folder of the project and another one inside the project subfolder. Delete both target folders since you are submitting the generated jar separately.
- You need to make sure your Part-B code can be compiled by entering *sbt clean assembly* command. We will run the compiled jar on our cluster directly using "spark-submit" with parameters. If your code cannot compile and run, you will not receive any points.

Assignment Tips!

- Please do not change the table name and its structure as provided in the assignment to keep it consistent.
- Please make sure that there is no syntax or other errors in your submission. In case of any syntax error, 0 marks will be given.
- For any case of doubt in the assignment, please make use of office hours and discussion board. Individual emails would not be entertained.
- Use PostgreSQL DBMS only. Use the version >= 13.