Assignment 2: Query Optimization in Spark SQL

Assigned: Thursday, January 27th, 2022
Due: Tuesday, February 22th, 2021 @ 11:59pm
Starter Code: https://github.com/stanford-futuredata/cs245-as2-public
Link to class Gradescope: https://www.gradescope.com/courses/344302

Overview

In this assignment, you will use Spark's SQL component to analyze query execution plans (Part I) and write some of your own query optimization rules (Part II). Part I involves running SQL queries in an interactive Spark shell, then writing up some analyses of the results. Part II involves writing Scala code to implement custom Spark SQL query optimization rules.

Setup

Software Dependencies

- Spark 2.3.3
  - Download Spark 2.3.3, prebuilt for Hadoop 2.7: https://archive.apache.org/dist/spark/spark-2.3.3/spark-2.3.3-bin-hadoop2.7.tgz.
  - This assignment requires Spark 2.3.3 specifically. Later versions of Spark perform too many optimizations automatically, making things a bit too easy :)
  - Running Spark requires Java 8 (unfortunately Java 9+ is not compatible). You must install Java 8 if you do not have it already. To check your Java version, run the command `java -version`. To install Java 8 on Ubuntu, run the command `sudo apt install openjdk-8-jdk`. On MacOS, follow the instructions here.
  - You must set the `JAVA_HOME` environment variable to your Java 8 JDK root directory. On Ubuntu, this will be `/usr/lib/jvm/java-1.8.0-openjdk-amd64`. On MacOS, to find this directory run the command `/usr/libexec/java_home -v 1.8`.

Background

We strongly suggest you read/review the following resources:

- The three lectures on Query Execution and Query Optimization
- The Spark SQL paper
  - (the assigned reading for week 5) http://web.stanford.edu/class/cs245/readings/spark-sql.pdf
- Spark SQL programming
  - The Overview and Getting Started sections in https://spark.apache.org/docs/2.3.3/sql-programming-guide.html
- Intro to Scala, if it's new to you
  - https://docs.scala-lang.org/tour/tour-of-scala.html

- https://learnxinyminutes.com/docs/scala/

Part I: Analyze Query Plans

- All of the material for Part I resides in the `part1/` folder in the starter code repository.
- Follow the instructions in `part1_handout.pdf`.
- When writing your solutions to the problems, you have two options:
  - 1. Use `LaTeX`, and write your solutions directly into `part1_handout.tex` (you should delete all the `\vspace{2in}` lines).
  - 2. Print out `part1_handout.pdf` and handwrite your solutions in the spaces we provide.

Part II: Write Your Own Optimization Rules

In this part, you will write a set of transforms to optimize Spark SQL logical plans that contain instances of a custom function (commonly known as a user-defined function, or UDF) we have defined called `dist`. This function computes the distance between two (x, y) points, and is defined as follows:

```
double dist(double x1, double y1, double x2, double y2)  {
  return sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2);
}
```

The code for this part is located at the top level of the assignment 2 repository. There are two main files: `Transforms.scala`, to which you will add your transforms, and `Tester.scala`, which runs several tests to check the correctness and completeness of your transforms. Feel free to add additional debugging code to `Tester.scala`, but you will only be submitting `Transforms.scala` on Gradescope and we will use an unmodified version of `Tester.scala` to grade your code.

Before you begin, run `mvn package` in the top level of the Assignment 2 repository. This command will take around 10 minutes the first time you run it, and will download all of the dependencies required for Spark SQL. You can then import the Assignment 2 repository into IntelliJ IDEA or your preferred IDE. Remember to import the top-level directory containing pom.xml. An IDE is recommended for this assignment so you can easily explore Apache Spark's code.

You must write four transforms for this assignment. The transforms take a Spark SQL `LogicalPlan` and produce an optimized `LogicalPlan`. In the following discussion, assume that we have a Spark SQL table T with four columns of type double: `x1`, `y1`, `x2`, and `y2`.

Your first three transforms will simplify expressions involving `dist` in any operator. We have provided an example transform of this type in `Transforms.scala`. Called `EliminateZeroDists`, it simplifies expressions of the form `dist(x, y, x, y)` to the constant value 0. The final transform you must implement is specific to the `Sort` operator.

1. Simplify comparison expressions involving a `dist` and a constant that always evaluate to either true or false. For example `select * from T where dist(x1, y1, x2, y2) > -1` should simplify to `select * from T` since `dist` must always be non-negative.
2. Replace any expression of the form `dist(...) == 0` with a simpler (and cheaper) expression involving just comparisons between the arguments to `dist`.
3. Replace comparisons between a `dist` and another `dist` or a constant value by squaring both sides to eliminate the square root computation. We have provided a helper method called `getDistSqUdf` to get an instance of the squared version of `dist`, called `dist_sq`. As an example of this transformation, `select * from T where dist(x1, y1, x2, y2) > 2` should become `select * from T where dist_sq(x1, y1, x2, y2) > 4`. (Note: due to floating point inaccuracies, this transform may not always be sound, but please ignore this for the purposes of the assignment.)
4. If an order expression in a `Sort` operator is a `dist`, replace it with `dist_sq` to avoid the square root computation, since the order of the squared values is the same as the order of the original values. (Note: when considering a `SortOrder` object, you can ignore its `sameOrderExpressions` field and focus on its `child` field.)

You must add all of the transforms you implement to the `Seq` returned by the `getOptimizationPasses` method in `Transforms.scala`. You may find it useful to add extra transforms beyond the ones we have asked for to simplify your implementation. One useful transform may be to reorder comparisons involving `dist` so that the `dist` is always on the left-hand side (or right-hand side) of the comparison. Note that the full list of transforms is run repeatedly until none of them make any changes to the logical plan ("run to fix point").

We have left in all of the necessary import statements in `Transforms.scala` to help guide your implementation. The IDE is your friend -- click through to the implementations of different Spark SQL classes (your IDE should allow you to download the Spark SQL sources) to see how they work and what arguments they expect.

To run the tests and check your score, edit the `run_tests.sh` script in the top level of the Assignment 2 repository and set the `SPARK_233_HOME` variable to point to the top level of your Spark 2.3.3 installation. Then run `./run_tests.sh`. To debug, you can look at the output of the above command to see which assertion is failing for a failed test, then look at your optimized logical plan (produced by the `explain(true)` method).

Note that we may make bug fixes to the tests before the assignment is due.

Grading

- Part I is worth a total of 70 points. Each of the 7 problems is worth 10 points. The one bonus problem (#8) is up to 5 points of extra credit, assigned based on the explanation.
- Part II is worth 140 points. There are 12 tests, with the first 10 worth 10 points each and the last two worth 20 points each.

Submission Instructions

- Submit a PDF (either generated from LaTeX or a scanned version of your handwritten solutions on the handout) to the gradescope assignment titled `Assignment 2: Part I (Written)` and please **tag** each question. **We will deduct 5 points** from any submitted solution that is not tagged properly.
- Submit your `Transforms.scala` to the Gradescope assignment titled `Assignment 2: Part II (Code)`.

Note that this assignment is to be done individually. We encourage students to form study groups, but all code must be written independently.

Notes

1) In case the following error occurs while running **mvn package**:

```
[ERROR] Failed to execute goal on project cs245-as2: Could not
resolve dependencies for project
edu.stanford:cs245-as2:jar:1.0-SNAPSHOT: Failed to collect
dependencies at org.apache.spark:spark-sql_2.11:jar:2.3.3 ->
org.apache.spark:spark-core_2.11:jar:2.3.3 ->
net.java.dev.jets3t:jets3t:jar:0.9.4 ->
commons-codec:commons-codec:jar:1.15-SNAPSHOT: Failed to read
artifact descriptor for
commons-codec:commons-codec:jar:1.15-SNAPSHOT: Could not
transfer artifact commons-codec:commons-codec:pom:1.15-SNAPSHOT
from/to maven-default-http-blocker (http://0.0.0.0/): Blocked
mirror for repositories: [scala-tools.org
(http://scala-tools.org/repo-releases, default,
releases+snapshots)] -> [Help 1]
```

   You can create a backup of **${maven.home}/conf/settings.xml** and remove the following entry from the settings.xml file:

```
<mirror> <id>maven-default-http-blocker</id>
<mirrorOf>external:http:*</mirrorOf> <name>Pseudo repository to mirror
external repositories initially using HTTP.</name>
<url>http://0.0.0.0/</url> </mirror>
```

   Maven home can be found by running the command **mvn –version.**

2) A starter guide for the assignment can be found at: https://youtu.be/lmG9Xlq0nTs