Assignment 4

Xiang Huang ZhongZheng Shu

Problem 1: Spatial Join

- Step 1

We instantiate a random instance in java to generate the points and rectangles. In order to meet the requirements, we generated 11000000 points and 8000000 rectangles.

- Step 2

In order to conduct an efficient spatial join, we have to partition the data to some cutomized grids and join the two datasets on these grids. First of all, we implemented a Composite Key Class called GridKey, which holds the joining key (Coordinates of customized grids) and an Intwritable representing the joining order. We override the original CompareTo and hashcode method to ensure that the key-value pairs are partitioned according to the joining key and sorted according to the joining order. Then we implemented two mapper classes for Rectangles and Points. In the corresponding mapper for rectangles, we split the original rectangle into several small rectangles that each falls in a specific grid, and output these rectangles with the rectangle name. As for points, we partition the point to their corresponding grids, and there exist a corner case (The point is on the corner of the grids) where one has to map that point to 4 grids around it. Then in the reduce phase, we just have to keep track of the rectangles in each specific grid and determine whether the following incoming records from points are within any of them.

There is a tradeoff between map phase and reduce phase. If you set the grid to be too large, it will foster the mapper phase by a little, but it will significantly slow down the reduce phase, as each time a new point record arrives, it has to go through all the rectangles within that grid. However, if you set the grid to be too small, it will generate many key-value pairs or a single rectangle, and it might lead to too much traffic in the sorting and partitioning phase. Therefore, we carefully choose the grid size to be height = 5 and width = 20, so as to ensure each record will generate at most 4 key-value pairs.

Problem 2: Customized Json Reader

- Step 1

  We created the dataset according to the requirements using the similar approach in problem 1. Here we generated 450000 customer records in Json format.

- Step 2

The key challenge in this problem lies in implementing a customized input format and a record reader. Here we used regex and LineRecordReader to implement out JsonRecordReader. It will start recording records when it finds a line starting with “{”, and return true after a “}” has been found. Our input format returns an instantiate of this class. The map reduce job is easy to write, as one just has to output Salary - Gender pairs in the mapper, and aggregate the records in the reducer.

Problem 3: K-means Algorithm

- Step 1

  We reused the points generated in problem 1 for clustering in this problem.

- Step 2

The implementation of K-means algorithm in Hadoop is broken down into two parts, a single map-reduce job and a controller for convergence. Here we used combiner and a single reducer in the map-reduce job, by mapping each input point to clusters according to their Euclidean Distance, and compute the new centroid in the reduce phase. We also keep track of the difference in centroids in the reducer and pass that value via the context of the job. Then in the outer controller, it manages the distributed cache which is used to store the centroid file, and check if the termination criteria has been met. When k = 4, our algorithm takes 19 iterations to converge.