Big Data Management Project 2

Spring 2017

Team 18 Caitlin Kuhlman, Wenlei Cao

**Code Organization:**

* **Project2**
  + **sjoin**
  + **json**
  + **kmeans**
* **lib**
* **project2.jar**

We have provided a precompiled jar called project2.jar. To compile the code yourself you must include the lib folder on the build path.

**Question 1. Spatial Join**

**To run:**

**Explanation:**

**Question 2. JsonInputFormat** Writing custom Input Format to read multi-line Json file.

**To run:**

hadoop jar <jarfile> json.Q2 <input path> <output path>

For example:

hadoop jar project2.jar json.Q2 /tmp/airfield.txt /tmp/output40

**Explanation:**

The package json contains 3 classes:

JsonRecordReader

JsonInputFormat

Q2

Q2 is the driver. It utilizes the other two classes to read the input file one json record at a time. The JsonInputFormat extends the FileInputFormat class and contains methods which define the RecordReader and the file split size.

The JsonRecordReader wraps a LineReader object which reads one line at a time from a text file. Extra logic is added to parse json objects, and keep track of where they start and end to process multiple input splits.

In Q2, the MRMapper class, extracts the flag number as a key and counter is included for each record as a value. In the MRReducer class, the counter is aggregated based on flag number.

To divide the file into 5 splits, we needed to tweak the maximum split size. This is done in the JsonInputFormat; we set the maximum size as 204800 (1/5 size of file).

**Question 3. Kmeans**

**To Run:**

hadoop jar <jarfile> kmeans.Q2 <input path> <centroid path> <output path> <k>

For example:

hadoop jar project2.jar kmeans.Q2 /input/points.csv /data/centroids output 50

**Explanation:**

The package kmeans contains 6 classes:

DataGenerator

KMeansMapper

KMeansCombiner

KMeansReducer

PointWritable

Q3

Q3 is the driver. It first generates k random points and puts them in a file on HDFS specified by the user. This file is sent via the distributed cache to mappers and reducers. It is reused in each iteration for the updated centroids.

The program runs up to 5 MapReduce jobs, stopping earlier only if the centroids have not changed from the previous iteration.

We include the 2 optimizations mentioned in the assignment: the use of a combiner class, and the use of a single reducer. In the reducer we check if the centroids have changed from the previous iteration and emit a message in the cleanup function to pass this information to the driver.

TODO: plan for spatial join

Use a Grid index to map points. This means since we know the input points all fall into a range of 1 -10,000, we use a grid to map them to sub areas (say a 100 x 100 grid each cell with a range of 100)

Every point gets mapped to a grid cell in one mapper

Every rectangle gets mapped to all the grid cells they overlap with in another mapper

Each reducer receives all points and rectangles that fall into one cell, and checks if any points fall within rectangles.

Checking if a point falls within a grid cell is easy:

point = (xp, yp)

grid cell = x from 0 to 100, y from 0 to 100 (for instance)

point falls in cell if:

0 <= xp < 100

and

0<=yp < 100

To check if a rectangle overlaps with a cell is more complicated - I guess like this: http://www.geeksforgeeks.org/find-two-rectangles-overlap/

Then to find if a point falls in a rectangle is easy too

point= (xp, yp)

rectangle= (xr, yr, h, w)

point falls in rectangle if:

xr <= xp < (xr+w)

and

yr<=yp < (y+h)