MapReduce Joins

|  |  |
| --- | --- |
| **Exercise Dir** | ~/workspace/map\_join |
| **Eclipse Proj** | map\_join |
| **JAR File** | mapside.jar |

1. Open the Project.

1.1. Right-click on the file NYSE\_dividends\_A\_join.csvand select “Open With-Text Editor”. You should recognize the dividends file from earlier labs. The data in this file will represent the “left” side of the join.

1.3. Similarly, open the file NYSE\_daily\_prices\_A.csv. This file is a subset of the “A” stocks. The data in this file is going to be joined by the Mapper to the dividend data.

1.4. In the project, open Stock.java. The Stock class encapsulates a stock symbol and the date it was traded. The hash map you use in the Mapper will store Stock objects as the key, and a double as its value.

1.5. Open StockPrices.java. The StockPrices class contains two doubles: a dividend and a closing price. The Mapper is going to output Stock objects keys and StockPrices objects as the values.

1.6. Open MapSideDriver.java. This class contains a Mapper class declaration named MapSideJoinMapper, and no Reducer.

2. Add the Dividend File as a LocalResource

2.1. The join is going to occur in the Mapper, and you are going to read in the entire contents of the dividends file before the Mapper starts mapping <key/value> pairs. In the run method, add the file NYSE\_dividends\_A\_join.csvas a LocalResource.

2.2. Also in the run method, set a configuration property named “stockSymbol” to the first command-line argument:

conf.set("stockSymbol", args[0]);

2.3. Notice the output format is already configured as TextOutputFormat, which uses a tab as the delimiter by default. We want the output to be a comma-delimited file, so add the following line of code in the run method:

conf.set(TextOutputFormat.SEPERATOR, ",");

3. Read in the Dividend File in the Mapper

3.1. A Mapper named MapSideJoinMapper is fixmebed out for you. Add a field of type String to MapSideJoinMapper:

private String stockSymbol;

3.2. Add a field of type Hashmap to the MapSideJoinMapper class:

private HashMap<Stock, Double> stocks = new HashMap<Stock, Double>();

NOTE: This HashMap is going to contain the stock and dividend price of every stock that matches the stockSymbol field.

3.3. In the setup method, initialize stockSymbol to the context parameter named “stockSymbol”.

3.4. In the setup method, use the getLocalCacheFiles method of Context to retrieve the files in the LocalResource (which returns an array, but there should only be one Path in the array).

3.5.The first file in the array should be the dividends file. Open this file using a FileSystem instance and wrap the stream into a BufferedReader instance.

3.6. Use the readLine method of BufferedReader to read in lines of text from the dividend file. The data coming in looks like:

NYSE,AIT,2009-11-12,0.15

3.7.If the stock symbol from the input text matches the stockSymbol field, then add the information to the HashMap by instantiating a new Stock object and initialize it with the stock symbol and date that you just read in from the dividends file. Use the put method of HashMap, with the Stock object as the key in the HashMap and the dividend amount is the value (which you will need to parse into a double).

3.8.The setup method is now complete. Note that after the setup method executes, all dividends will have been processed and the HashMap will contain all of the dividend amounts (along with the dates) of a single stock.

4.Join the Dividends with the Stock Prices

4.1. The data coming in to the Mapper looks like:

NYSE,AIT,2010-02-08,21.81,21.81,21.28,21.31,103800,21.31

In the map method of MapSideJoinMapper, split the incoming line of text into an array of String objects using a comma as the delimiter.

4.2. If the stock symbol being processed does not match the stockSymbol field, then do not do anything in the map method this line of input can be ignored.

4.3. If the stock symbol matches the stockSymbol field, then use the stock symbol and date of the input to create a new Stock object.

4.4. Use the containsKey method of the HashMap to see if your new Stock object is in the HashMap.

4.5. If containsKey returns false, then ignore this line of input.

4.6. If containsKey returns true, then this current line of input needs to be joined with the corresponding value in the HashMap. Start by using the get method of HashMap to retrieve the dividend.

4.7. Using the dividend from the HashMap and the closing price (index number 6 in the array of input String objects), create a new StockPrices object.

4.8. Output the Stock object as the key and the StockPrices object as the value.

4.9. Save your changes to MapSideDriver.java.

5. Run the Job

5.1. Build the project to create mapside.jar.

5.2. Put the “A” stocks into HDFS:

$ cd ~/data/  
 $ hdfs dfs -put stocks/\* stocks

5.3. Run the MapSideJoin application on the AIT stock:

$ yarn jar mapsidejoin.jar AIT

5.4. You should see a new folder for the output named joinoutputthat contains the output of the job. For verification, if you run the job on the AIT stock, the output should look like the following:

AIT,2009-11-12,0.15,20.65  
 AIT,2009-08-12,0.15,22.18  
 AIT,2009-05-13,0.15,19.75  
 AIT,2009-02-11,0.15,17.23  
 ...and so on

Result: Your application joins the data in the dividends file with the data in the stock prices file. The result is a list of dates where a dividend was paid, along with the closing price of the stock on that day. The join is accomplished by reading in the entire dividends file before the Mapper starts and sharing the file as a LocalResource. The Mapper then reads through all of the stock prices, searching for input that matches data in the dividends file. This design works well when one of the input files is small enough to be copied into the memory of each Mapper.

**END**