Shayideep Sangam

Georgia state university – Fall 2018

ASSIGNMENT 8

In BDP-11-frequent items-1, we have discussed a-priori algorithm to identify frequent pairs. The Spark implementation we discussed has some flaw: we loop through all the elements contained by a RDD, which is not a good solution since RDD is a distributed collection of data elements, across different worker nodes in the cluster. In general, we should use transformations to specify what we would like to do for *each* element contained by the RDD, and let the cluster to manage the rest, we should never explicitly loop over its elements.

In this assignment, you will need to fix this issue. The details can be found in slide 42 – 45 from BDP-11-frequent items-1. As always, you can find our code discussed in the class from our course web site (the related text file is called `baskets.txt`). In the code, look for this comment, `===> the following has to be changed`, so you know where to replace/fix. Furthermore, *see slide 44 and slide 50 for hints*.

SOLUTION:

1. *all* your source code, with necessary comments so I know what is the part you have changed/fixed (90 points)

```
import java.util.*;


import org.apache.spark.mllib.fpm.AssociationRules;

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.api.java.function.FlatMapFunction;

import org.apache.spark.api.java.function.Function;

import org.apache.spark.api.java.function.Function2;

import org.apache.spark.api.java.function.PairFlatMapFunction;

import org.apache.spark.api.java.function.PairFunction;

import org.apache.spark.api.java.function.VoidFunction;
```

```
import org.apache.spark.broadcast.Broadcast;
import org.apache.spark.sql.SparkSession;

import scala.Tuple2;

// this is the example in Chap 3, Example 3.6

public class SparkFrequency {

    private static final String FILE_URI = "C:/Users/Avinash/Downloads/baskets.txt";
    private static final double s = 0.5; // threshold = 50%

    public static void main(String[] args) {

        // initializing spark
        SparkSession spark = SparkSession.builder().config("spark.master", "local[*]").getOrCreate();
        JavaSparkContext sc = new JavaSparkContext(spark.sparkContext());
        sc.setLogLevel("WARN");

        // create RDD by using text files
        JavaRDD<String> baskets = sc.textFile(FILE_URI);
        System.out.println(baskets.take((int) baskets.count()).toString());

        // total number of baskets
        Broadcast<Long> basketCount = sc.broadcast(baskets.count());

        // organize basket content into integer array
        JavaRDD<Integer[]> basketContent = baskets.map(new Function<String, Integer[]>() {
            public Integer[] call(String line) throws Exception {
                String[] itemStr = line.substring(1, line.length() - 1).split(","); // get rid of { and }, and split
                Integer[] items = new Integer[itemStr.length];
```

```
            for (int i = 0; i < itemStr.length; i++) {
                items[i] = new Integer(Integer.parseInt(itemStr[i].trim()));
            }
            return items;
        }
    });
    System.out.println("basketContent has [" + basketContent.count() + "] elements");
    basketContent.foreach(new VoidFunction<Integer[]>() {
        public void call(Integer[] items) throws Exception {
            for (int i = 0; i < items.length; i++) {
                System.out.print(items[i] + " ");
            }
            System.out.println();
        }
    });

    // first pass: list all the items
    JavaPairRDD<Integer, Integer> items = basketContent.flatMap(new FlatMapFunction<Integer[],
Integer>() {
        public Iterator<Integer> call(Integer[] items) throws Exception {
            return Arrays.asList(items).iterator();
        }
    }).mapToPair(new PairFunction<Integer, Integer, Integer>() {
        public Tuple2<Integer, Integer> call(Integer item) throws Exception {
            return new Tuple2<Integer, Integer>(item, 1);
        }
    });
    System.out.println("items has [" + items.count() + "] elements");
    System.out.println(items.take((int) items.count()).toString());

    // first pass: count each item
```

```
JavaPairRDD<Integer, Integer> itemCounts = items.reduceByKey(new Function2<Integer, Integer, Integer>() {
    public Integer call(final Integer value1, final Integer value2) {
        return value1 + value2;
    }
}).sortByKey();
System.out.println("itemCounts has [" + itemCounts.count() + "] elements");
System.out.println(itemCounts.take((int) itemCounts.count()).toString());

// first pass: create frequent-items table
// ===> the following has to be changed since this is not the best way to do this (see slide 43 of BDP-11)
Broadcast<Double> sValue = sc.broadcast(s);
int[] frequentItems = new int[(int) (itemCounts.count() + 1)];
int threshold = (int) Math.ceil(s * basketCount.value());


//      filter item count to to get frequent items
Map<Integer, Integer> collect = itemCounts.filter(new Function<Tuple2<Integer, Integer>, Boolean>() {
    @Override
    public Boolean call(Tuple2<Integer, Integer> integerIntegerTuple2) throws Exception {
        return integerIntegerTuple2._2 >= threshold;
    }
}).collectAsMap();
//      Collect the Frequent items
for (int i = 1; i <= itemCounts.count(); i++) {
    frequentItems[i] = collect.containsKey(i) ? collect.get(i) : 0;
}
```

```
// broadcast the table to avoid network traffic
// the size should be fine
System.out.println("frequentItems[i]:");
for (int i = 1; i < frequentItems.length; i++) {
   System.out.print("count[" + i + "] = " + frequentItems[i]);
   if (i < frequentItems.length - 1) {
      System.out.print(", ");
   } else System.out.println();
}
Broadcast<int[]> frequentItemTable = sc.broadcast(frequentItems);
// ===> end of the changed part

// second pass: generate frequent-pairs, start from basketContent
class FrequentPairsChecker implements PairFlatMapFunction<Integer[], String, Integer> {

   public Iterator<Tuple2<String, Integer>> call(Integer[] items) throws Exception {

      List<Tuple2<String, Integer>> frequentPairs = new ArrayList<>();

      int[] intItems = Arrays.stream(items).mapToInt(Integer::intValue).toArray();
      String itemStr = SparkAPrioriUtils.generatePairs(intItems);

      if (itemStr == null) return frequentPairs.iterator();

      String[] itemPairs = itemStr.split(",");
      for (String itemPair : itemPairs) {
         String[] tmpPair = itemPair.split("-");
         if (tmpPair == null || tmpPair.length != 2) continue;
         if (Integer.parseInt(tmpPair[0]) > Integer.parseInt(tmpPair[1])) {
            String tmpStr = tmpPair[0];
            tmpPair[0] = tmpPair[1];
```

```
            tmpPair[1] = tmpStr;
        }  // make sure the item pair is ordered, such as 123-234, not the other way around

        // check frequent-items table, to make sure both are frequent
        if (frequentItemTable.value()[Integer.parseInt(tmpPair[0])] > 0 &&
            frequentItemTable.value()[Integer.parseInt(tmpPair[1])] > 0) {
            frequentPairs.add(new Tuple2<>(tmpPair[0] + "-" + tmpPair[1], 1));
        }

    }

    return frequentPairs.iterator();
  }
}

JavaPairRDD<String, Integer> candidateDoubles = basketContent.flatMapToPair(new
FrequentPairsChecker());
System.out.println("candidateDoubles has [" + candidateDoubles.count() + "] elements");
System.out.println(candidateDoubles.take((int) candidateDoubles.count()).toString());

JavaPairRDD<String, Integer> frequentPairs =
    candidateDoubles.reduceByKey(new Function2<Integer, Integer, Integer>() {
        public Integer call(final Integer value1, final Integer value2) {
            return value1 + value2;
        }
    }).filter(new Function<Tuple2<String, Integer>, Boolean>() {
        public Boolean call(Tuple2<String, Integer> frequentPair) {
            if (frequentPair._2 >= (int) Math.ceil(sValue.value() * basketCount.value()))
                return true;
            else return false;
        }
```

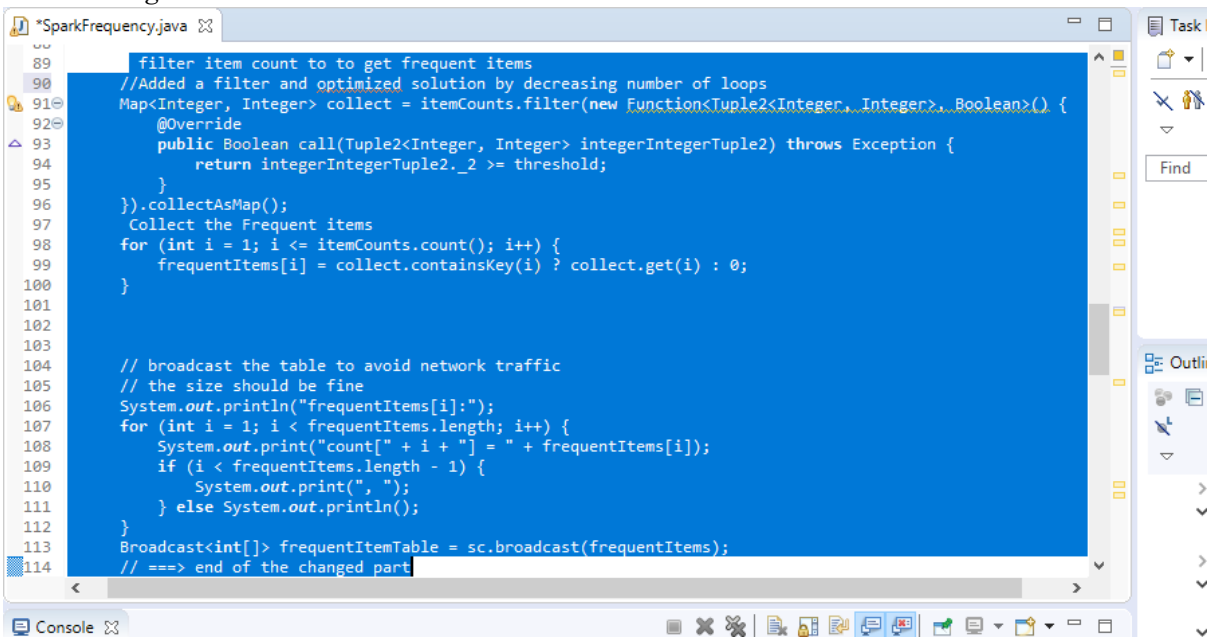
```
        });
    System.out.println("final frequent pairs =>");
    System.out.println(frequentPairs.take((int) frequentPairs.count()).toString());

    frequentItemTable.unpersist();
    frequentItemTable.destroy();
    basketCount.unpersist();
    basketCount.destroy();

    sc.close();
  }

}
```

1. Code changes screen shot



2. an *acceptable* screen copy to show the results of running your code (10 points)

eclipse-workspace - SparkApriori/src/SparkFrequency.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- HadoopWordCount
- matrix123
- pagerank
- spark5
  - JRE System Library [JavaSE-1.8]
  - src
    - (default package)
      - MinHashHelper.java
      - MinHashUtils.java
      - ShingleUtils.java
      - SparkMinHashLSH.java
  - Referenced Libraries
- SparkApriori
  - JRE System Library [JavaSE-1.8]
  - src
    - (default package)
      - SparkAPrioriUtils.java
      - SparkFrequency.java
  - Referenced Libraries
- Sparkpagerank
- SparkRecSys
  - JRE System Library [JavaSE-1.8]
  - src
    - (default package)
      - SparkRecommendationSystemDocument.java
  - Referenced Libraries

SparkFrequency.java

```
1  import java.util.*;
17
18 // this is the example in Chap 3, Example 3.6
```

Console

<terminated> SparkFrequency [Java Application] C:\Program Files\Java\jdk1.8.0_91\bin\javaw.exe (19 Nov 2018, 22:29:52)

```
2018-11-19 22:29:57 INFO  BlockManagerMaster:54 - Registered BlockManager BlockManagerId(driver, Deep, 50199, None)
2018-11-19 22:29:57 INFO  BlockManager:54 - Initialized BlockManager: BlockManagerId(driver, Deep, 50199, None)
2018-11-19 22:29:57 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@3a43d133{/metrics/json,null,AVAI
[{9, 17,2,3,7}, {17,16,15,14,13,12,11,10}, {10,15,6,5,9}, {15,17,16,4,11,10,8,1}, {17}, {15,17,16,9,8,11,7}]
basketContent has [6] elements
15 17 16 4 11 10 8 1
17
15 17 16 9 8 11 7
9 17 2 3 7
17 16 15 14 13 12 11 10
10 15 6 5 9
items has [34] elements
[(9,1), (17,1), (2,1), (3,1), (7,1), (17,1), (16,1), (15,1), (14,1), (13,1), (12,1), (11,1), (10,1), (10,1), (15,1),
itemCounts has [17] elements
[(1,1), (2,1), (3,1), (4,1), (5,1), (6,1), (7,2), (8,2), (9,3), (10,3), (11,3), (12,1), (13,1), (14,1), (15,4), (16,3
frequentItems[i]:
count[1] = 0, count[2] = 0, count[3] = 0, count[4] = 0, count[5] = 0, count[6] = 0, count[7] = 0, count[8] = 0, count
candidateDoubles has [34] elements
[(9-17,1), (16-17,1), (15-17,1), (11-17,1), (10-17,1), (15-16,1), (11-16,1), (10-16,1), (11-15,1), (10-15,1), (10-11,
final frequent pairs =>
[(11-16,3), (16-17,3), (15-16,3), (10-15,3), (11-17,3), (11-15,3), (15-17,3)]
```

Task List

Find All Ac...

Outline

- SparkFrequency
  - FILE_URI : String
  - s : double
  - main(String[]) :
    - new Functio
      - call(String
    - new VoidFun

Javadoc Declaration

long org.apache.spark.api.java.AbstractJavaRDDLike.count()