Big data: architectures and data analytics

Exam: February 5, 2021 - Big data: architectures and data analytics

	FEIHONG SHI 274816
Iniziato	venerdì, 5 febbraio 2021, 08:06
Terminato	venerdì, 5 febbraio 2021, 09:57
Tempo impiegato	1 ora 51 min.
Valutazione	**27,00** su un massimo di 31,00 (**87**%)

Domanda 1

Risposta corretta

Punteggio ottenuto 2,00 su 2,00

Consider the input HDFS folder myFolder that contains the following two files:

- NamesItaly.txt
  - The text file NamesItaly.txt contains the following three lines (size: 33 bytes)

    Luis,Turin
    Luca,Rome
    Paolo,Turin
- NamesFrance.txt
  - The text file NamesFrance.txt contains the following two lines (size: 22 bytes)

    Paolo,Nice
    Luis,Paris

Suppose that you are using a Hadoop cluster that can potentially run up to 5 instances of the mapper class in parallel. Suppose the HDFS block size is 128MB. Suppose to execute a MapReduce application for Hadoop that analyzes the content of myFolder. Suppose the map phase emits overall the following 5 key-value pairs (the key part is a name while the value part is always 1):

(“Luis”, 1)
(“Luca”, 1)
(“Paolo”, 1)
(“Paolo”, 1)
(“Luis”, 1)

Suppose the number of instances of the reducer class is set to 4 and suppose the reduce method of the reducer class sums the values associated with each key and emits one pair (name, sum values) for the keys associated with "sum values" greater than or equal to 2. Suppose the following 2 pairs are overall emitted by the reduce phase:

(“Paolo”, 2)
(“Luis”, 2)

Considering all the instances of the reducer class, overall, how many times is the **reduce method invoked**?

- (a) 2
- (b) 4
- (c) 3 ✓
- (d) 5

Risposta corretta.

La risposta corretta è: 3

Domanda 2

Risposta corretta

Punteggio ottenuto 2,00 su 2,00

Consider the input HDFS folder myFolder that contains the following two files:

- Temperatures2019.txt: size = 1030MB
- Temperatures2020.txt: size = 1018MB

Suppose that you are using a Hadoop cluster that can potentially run up to 10 instances of the mapper class in parallel. Suppose to execute a MapReduce application for Hadoop that computes some statistics about temperatures. This application is based on one single MapReduce job. The input of this Hadoop application is the HDFS folder myFolder. The HDFS block size is 1024MB. The number of instances of the reducer class is set to 5.

How many mappers are instantiated by Hadoop (i.e., how many instances of the mapper class) when you execute this MapReduce application by specifying the folder myFolder as input?

- (a) 5
- (b) 10
- (c) 2
- (d) 3 ✓

Risposta corretta.

La risposta corretta è: 3

Domanda 3

Completo

Punteggio ottenuto 4,00 su 7,00

PoliCars is an international company characterized by several production plants around the world. In each production plant, robots are used to produce some components of the PoliCars's cars. PoliCars computes a set of statistics about its production plants and robots. The analyses are performed by considering the following input data sets/files related to the failures of the PoliCars's robots.

- ProductionPlants.txt
  - ProductionPlants.txt is a text file containing the list of production plants of PoliCars. Each line of ProductionPlants.txt is associated with one production plant. The number of

production plants is 1000. In some cities, there is more than one production plant.
    - Each line of ProductionPlants.txt has the following format
        - PlantID,City,Country
        - where PlantID is the production plant identifier while City and Country are the city and the country in which the production plant is located, respectively.
    - For example, the following line
        - PID1,Turin,Italy
        - means that the production plant PID1 is located in Turin, Italy

- Robots.txt
    - Robots.txt is a text file containing the list of robots managed by PoliCars. One line for each robot of PoliCars is stored in Robots.txt. The number of managed robots is more than 15000.
    - Each line of Robots.txt has the following format
        - RID,PlantID,IP
        - where RID is the robot identifier, PlantID is the identifier of the production plant in which the robot is installed, and IP is its IP address.
    - For example, the following line
        - R15,PID1,130.192.20.21
        - means that robot R15 is installed in the production plant PID1 and the IP address associated with that robot is 130.192.20.21.

- Failures.txt
    - Failures.txt containing the historical information about the failures of the robots. A new line is inserted in Failures.txt every time there is a failure of a robot. The number of managed robots is more than 15000 and Failures.txt contains the historical data about the Failures of the last 20 years.
    - Each line of Failures.txt has the following format
        - RID,FailureTypeCode,Date,Time
        - where RID is the identifier of the robot that had a failure, Date is the date of the failure, Time is the time of the failure, and FailureTypeCode is the code of the type of Failure.
    - For example, the following line
        - R15,FCode122,2020/05/01,06:40:51
        - means that robot R15 had a failure of type FCode122 at 06:40:51 (hour=06, minute=40, second=51) of May 1, 2020.

The managers of PoliCars are interested in performing some analyses about their robots.

Design a single application, based on MapReduce and Hadoop, and write the corresponding Java code, to address the following point:

1. *Number of robots associated with the failure type FCode122 at least one time*. The application computes the number of robots associated with the failure type FCode122 at least one time. Store the computed value in the output HDFS folder (the output will contain one single line associated with the computed value).

Example

For instance, suppose that Failures.txt contains only the following lines:

R1,FCode122,2020/05/01,06:40:51

R1,FCode36,2020/05/02,06:40:51

R1,FCode122,2020/05/03,06:40:51

R2,FCode122,2020/05/04,06:40:51

R2,FCode11,2020/05/05,06:40:51

R3,FCode54,2020/05/14,06:40:51

In this case the number of robots associated with the failure type FCode122 at least one time is two and hence one single line containing the value 2 is stored in the output folder.

Suppose that the input is Failures.txt and it has been already set and also the name of the output folder has been already set.

- Write only the content of the Mapper and Reducer classes (map and reduce methods. setup and cleanup if needed). The content of the Driver must not be reported.
- Use the next two specific multiple-choice questions to specify the number of instances of the reducer class for each job.
- If your application is based on two jobs, specify which methods are associated with the first job and which are associated with the second job.
- If you need personalized classes report for each of them:
  - name of the class
  - attributes/fields of the class (data type and name)
  - personalized methods (if any), e.g, the content of the toString() method if you override it
  - do not report get and set methods. I suppose they are "automatically defined"

Job #1

/* Mapper */

```
class MapperBigData extends Mapper<
                LongWritable, // Input key type
                Text,         // Input value type
                Text,         // Output key type - set the appropriate data type
                Text> {// Output value type - set the appropriate data type


   protected void map(
         LongWritable key,   // Input key type
         Text value,         // Input value type
         Context context) throws IOException, InterruptedException {
      /* INSERT YOUR CODE HERE */
      String [] fields = value.toString().split(",");
```

```
        String FailType = fields[1];
        String RID = fileds[0];
        if(FailType.compareTo("FCode122")==0){
            context.write(new Text(FailType), new Text(RID));
        };

    }
}
```

```
/* Reducer */
/* REMOVE THIS PART IF YOUR JOB IS A MAP ONLY JOB */
class ReducerBigData extends
        Reducer<Text, // Input key type - set the appropriate data type
                Text, // Input value type - set the appropriate data type
                Text, // Output key type - set the appropriate data type
                IntWritable> { // Output value type - set the appropriate data type

    protected void reduce(Text key, // Input key type
            Iterable<Text> values, // Input value type
            Context context) throws IOException, InterruptedException {
        /* INSERT YOUR CODE HERE */
        int numOfRobot = 1;
        String preRID = null;
        Arrary.sort(values);
        for(String RID : values){
            if(preRID != null && preRID != RID){
                numOfRobot = numOfRobot + 1;
            };
            preRID = RID;
        };
        context.write(key, new IntWriteable(numOfRobot));
    }
}
```

Commento:
The computation of the distinct number of RIDs in the reduce method is not correct. -3 points

Domanda 4

Risposta corretta

Punteggio ottenuto 0,50 su 0,50

Exercise MapReduce - Number of instances of the reducer - Job 1

Select the number of instances of the reducer class of the first Job

- (a) 0
- (b) exactly 1 ✓
- (c) any number >=1

Risposta corretta.

La risposta corretta è: exactly 1

Domanda 5

Risposta corretta

Punteggio ottenuto 0,50 su 0,50

Exercise MapReduce - Number of instances of the reducer - Job 2

Select the number of instances of the reducer class of the second Job

- (a) One single job is needed for this MapReduce application ✓
- (b) 0
- (c) exactly 1
- (d) any number >=1

Risposta corretta.

La risposta corretta è: One single job is needed for this MapReduce application

Domanda 6

Completo

Punteggio ottenuto 18,00 su 19,00

Exercise Spark

PoliCars is an international company characterized by several production plants around the world. In each production plant, robots are used to produce some components of the PoliCars's cars. PoliCars computes a set of statistics about its production plants and robots. The analyses are performed by considering the following input data sets/files related to the failures of the PoliCars's robots.

- ProductionPlants.txt
  - ProductionPlants.txt is a text file containing the list of production plants of PoliCars. Each line of ProductionPlants.txt is associated with one production plant. The number of production plants is 1000. In some cities, there is more than one production plant.
  - Each line of ProductionPlants.txt has the following format
    - PlantID,City,Country
    - where PlantID is the production plant identifier while City and Country are the city and the country in which the production plant is located, respectively.
  - For example, the following line

        - PID1,Turin,Italy
        - means that the production plant PID1 is located in Turin, Italy

- Robots.txt
    - Robots.txt is a text file containing the list of robots managed by PoliCars. One line for each robot of PoliCars is stored in Robots.txt. The number of managed robots is more than 15000.
    - Each line of Robots.txt has the following format
        - RID,PlantID,IP
        - where RID is the robot identifier, PlantID is the identifier of the production plant in which the robot is installed, and IP is its IP address.
    - For example, the following line
        - R15,PID1,130.192.20.21
        - means that robot R15 is installed in the production plant PID1 and the IP address associated with that robot is 130.192.20.21.

- Failures.txt
    - Failures.txt containing the historical information about the failures of the robots. A new line is inserted in Failures.txt every time there is a failure of a robot. The number of managed robots is more than 15000 and Failures.txt contains the historical data about the Failures of the last 20 years.
    - Each line of Failures.txt has the following format
        - RID,FailureTypeCode,Date,Time
        - where RID is the identifier of the robot that had a failure, Date is the date of the failure, Time is the time of the failure, and FailureTypeCode is the code of the type of Failure.
    - For example, the following line
        - R15,FCode122,2020/05/01,06:40:51
        - means that robot R15 had a failure of type FCode122 at 06:40:51 (hour=06, minute=40, second=51) of May 1, 2020.

The managers of PoliCars are interested in performing some analyses related to their production plants and robots.

The managers of PoliCars asked you to develop one single application to address all the analyses they are interested in. The application has five arguments: the three input files ProductionPlants.txt, Robots.txt, Failures.txt and two output folders, “outPart1/” and “outPart2/”, which are associated with the outputs of the following Points 1 and 2, respectively.

Specifically, design a single application, based on Spark, and write the corresponding code, to address the following points:

1. Production plants with at least one robot with at least 50 failures in year 2020. The application considers only the failures occurred in year 2020 and selects the identifiers (PlantIDs) of the production plants with at least one robot associated with at least 50 failures in year 2020. The PlantIDs of the selected production plants are stored in the first HDFS output folder (one PlantID per line).

2. For each production plant compute the number of robots with at least one failure in year 2020. The application considers only the failures occurred in year 2020 and computes for each production plant the number of its robots associated with at least one failure in year 2020. The application stores in the second HDFS output folder, for each production plant, its PlantID and the computed number of robots with at least one failure in year 2020.
   Pay attention that the output folder must contain also one line for each of the production plants for which all robots had no failures in year 2020. For those production plants the associated output line is (PlantID, 0).

Examples Point 2

- Suppose that in the production plant PID1 there are three robots identified by the identifiers R1, R2, and R3, respectively. Suppose that the number of failures in year 2020 of R1 is 10, the number of failures in year 2020 of R2 is 15, and the number of failures in year 2020 of R3 is 3. In this case, in the second HDFS output folder, the output line associated with the production plant PID1 will be
  - PID1,3

- Suppose that in the production plant PID2 there are two robots identified by the identifiers R4 and R5, respectively. Suppose that the number of failures in year 2020 of R4 is 0 and the number of failures in year 2020 of R5 is also 0. In this case, in the second HDFS output folder, the output line associated with the production plant PID2 will be
  - PID2,0

If you need personalized classes report for each of them:

- name of the class
- attributes/fields of the class (data type and name)
- personalized methods (if any), e.g, the content of the toString() method if you override it
- do not report get and set methods. I suppose they are "automatically defined"

```
...
/* Suppose all the needed imports are already set */
...
public class SparkDriver {

   public static void main(String[] args) {
      String inProdPlants;
      String inRobots;
      String inProdPlants;
      String outputPathPart1;
      String outputPathPart2;
      inProdPlants = "ProductionPlants.txt";
      inRobots = "Robots.txt";
```

```
        inFailures = "Failures.txt";

        outputPathPart1 = "outPart1/";
        outputPathPart2 = "outPart2/";
        // Create a configuration object and set the name of the application
        SparkConf conf = new SparkConf().setAppName("Spark Exam -
Exercise #2");
        // Create a Spark Context object
        JavaSparkContext sc = new JavaSparkContext(conf);

       /* Write your code here */
      // Part 1
      // inFailures

      JavaRDD<String> failures = sc.textFile(inFailures).cache();
      JavaRDD<String> failures2020 = failures.filter(line -> {
         String[] fields = line.split(",");
         String year = fields[2];
         if(year.startWith("2020"))
            return true;
         else
            return false;
       });

      JavaPairRDD<String, Integer> failures2020Map = failures2020.mapToPair(line ->{
         String[] fields = line.split(",");
         String rid = fields[0];
         return new Tuple2<String, Integer> (rid, 1);
      });

      JavaPairRDD<String, Integer> failures2020Reduce = failures2020Map.reduceByKey((v1, v2)
      -> v1+v2);

      JavaPairRDD<String, Integer> failures2020filter50 = failures2020Reduce.filter(line -> line._2()
      >= 50);

      // inRobots
      JavaRDD<String> robots = sc.textFile(inRobots).cache();

      JavaPairRDD<String, String> robotsPair = robots.mapToPair(line -> {
         String[] fields = line.split(",");
         String rid = fields[0];
         String pid = fields[1];
         return new Tuple2<String, String> (rid, pid)
      });
```

```
JavaPairRDD <String, Tuple2<Integer, String>> joinFailureRobot =
failures2020filter50.join(robotsPair);

JavaRDD<String, Integer> part1Result = joinFailureRobot.mapToPair(line -> {
   pid = line._2()._2();
   return new Tuple2<String, Integer>(pid, 1);
});

JavaPairRDD<String, Integer> part1ResultSum = part1Result.reduceByKey((v1, v2) ->
v1+v2);
JavaRDD <String> part1ResultFinal = part1ResultSum.map(line -> {
   String pid = line._1();
   return pid;
});

part1ResultFinal.saveAsTextFile(outputPathPart1);

// Part2
JavaRDD<String> getPids = sc.textFile(inProdPlants).cache();

JavaPairRDD<String, Integer> initPids = getPids.mapToPair(line -> {
   String[] fields = line.split(",");
   String pid = fields[0];
   return new Tuple2<String, Integer>(pid, 0);
});

JavaPairRDD<String, Tuple<Integer, String> > getFailuresRobots =
failures2020Reduce.join(robotsPair);

JavaPairRDD<String, Integer> changeKeyToPid = getFailuresRobots.mapToPair( line -> {
   String pid = line._2()._2();
   return new Tuple2<String, Integer>(pid, 1);
});

JavaPairRDD<String, Integer> getRobotsEachPid = changeKeyToPid.reduceByKey((v1, v2) -
> v1+v2);

JavaPairRDD<String, Integer> getNullRobotsEachPid =
initPids.subtractByKey(getRobotsEachPid);

// I do not know this kind wirte is right or not, but the idea is union the two
RDDs(getNullRobotsEachPid, getRobotsEachPid) into one RDDs, so that it can be
// output as one file.
JavaPairRDD<String, Integer> part2Result =
getNullRobotsEachPid.union(getRobotsEachPid);
```

```
        part2Result.saveAsTextFile(outputPathPart2);
    }
```

Commento:
Part 1.
You can use map + distinct to select the distinct plantIDs. You do not need mapToPair + reduceByKey. -1 point