Autumn 2017

Big Data Management

Hands-on MapReduce programming exercises

Department of Computer Science
University of Helsinki

General instruction: You may discuss the questions with your classmates, but all the codes and analysis should be done independently. The points will be given based on the correctness and the running time of your programs. Therefore, try to find all correct results and reduce the running time of your programs.

1. Implement three executable Hadoop MapReduce programs to perform the inner join of two tables based on "*Student ID"* to satisfy the following two filtering conditions simultaneously:
   a). The year of birth is greater than (>) 1990 and
   b). The score of course 1 is greater than (>) 80 and that of course 2 is no more than (<=) 95.

   Sample data of Student table:

Student ID	Name	Year of Birth
20170126453	Kristalee Copperwaite	2000
20170433596	Roeberta Naden	1997

   Sample data of Score table:

Student ID	Score for course 1	Score for course 2	Score for course3
20170126453	93	97	80
20170140241	86	85	87
20170433596	82	60	80

   Join results:

Student ID	Name	Year of Birth	Score for course 1	Score for course 2	Score for course3
20170433596	Roeberta Naden	1997	82	60	80

Download datasets: https://www.cs.helsinki.fi/u/jilu/dataset/TableJoin_DataSet.zip
There are four files in the unzipped folder: three *Score* tables and one *Student* table.
Considering the following three cases, design three **different** MapReduce programs to efficiently handle them.

1. Both *Student* table and *Score* table are big tables with thirty million entries. That is, "*score_30000000.txt*" and "*student_30000000.txt*".
2. *Student* table is a big table, but *Score* table is small with three thousand entries. That is, "*score_3000.txt*" and "*student_30000000.txt*". (Hint: Use the distributed cache in this case)
3. *Student* table is still the same one, but *Score* table is a medium-size table with three million entries. That is, "*score_3000000.txt*" and "*student_30000000.txt*". (Hint: Use bloom filter in this case)

Run your three different MapReduce programs to each combination of datasets, and then report the result size and the performance, e.g. the elapsed time by filling the following two tables.

(Hint: you may get the job information and elapsed time through the web interface of master node, default address is hostname:8088 in Hadoop cluster)

Result size (10 points):

Data combination	Result size
"score_30000000.txt" and "student_30000000.txt"	
"score_3000.txt" and "student_30000000.txt"	
"score_3000000.txt" and "student_30000000.txt"	

Note that if your three MapReduce programs return the different result size for the same datasets, please check your codes carefully and make sure that their answer sizes are consistent.

Total running time (ms) (10 points)

Data combination	Program (1)	Program (2)	Program (3)
"score_30000000.txt" and "student_30000000.txt"			
"score_3000.txt" and "student_30000000.txt"			
"score_3000000.txt" and "student_30000000.txt"			

Questions: Analyze your results by answering the following questions:

a. How many computer nodes did you use to run the program? Have you tried to reduce the running time by using more nodes in Ukko cluster? (5 points)
b. Upload your source codes and analyze the performance of your codes. What optimization have you tried to reduce the running time for the large dataset? (20 points)
c. What is the elapsed time for the construction of bloom filter in Program 3? (5 points)

2. Implement Hadoop MapReduce programs to perform the similarity string join with edit distance. Download datasets: https://www.cs.helsinki.fi/u/jilu/dataset/EditDistance.zip, where you may find two folders including four files. Each file in the "small dataset" has 10k lines, while the "large" one contains 25M lines. The similarity join returns all pairs of strings (S1, S2) from two files, such that their edit distance of S1 and S2 is <= θ. For example, two strings "*abcd*" and "*abce*" are the similar pair with edit distance θ=1.

Edit distance definition: Given two strings S1 and S2 on an alphabet Σ, the edit distance ED(S1, S2) is the minimum number series of edit operations that transform S1 into S2. There are three operations which can be supported by the edit distance:

- Insertion of a single symbol. If S = "uv", then inserting the symbol "x" produces "uxv".
- Deletion of a single symbol changes "uxv" to "uv".
- Substitution of a single symbol "x" for a symbol "y" ≠ "x" changes "uxv" to "uyv".

Run your MapReduce programs to perform similarity join for two files in the same folder (i.e. join two big files and two small files) with two thresholds (θ=1, or θ=2). That is, you need to run similarity joins four times with two datasets and two thresholds each, and then show the result size and the performance by filling the following table.

Total running time and result size (ms) (15 points)

Data combination	Join threshold θ	Result size	Running time (ms)
Small dataset	1	3	
	2	7	
Big dataset	1		
	2		

Hint 1: You may use the small dataset to test the correctness of your algorithm. The correct answers are already shown in the table.

Hint 2: You may feed the output of one Reducer directly to another Mapper by using the following lines:

```
job1.setOutputFormatClass(SequenceFileOutputFormat.class)
```

and

```
job2.setInputFormatClass(SequenceFileInputFormat.class)
```

Questions: Analyze your results by answering the following questions:

a. Describe your algorithms for this problem. Note that the brute-force implementation to compare the edit distance of every two strings may not work for the large dataset. (15 points)

b. Upload your source codes and analyze the performance of your codes. Why is the edit distance join slower than the algorithm in the 1st exercise? (20 points)

Reference for your programming:

- Miner D, Shook A. MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems [M]. " O'Reilly Media, Inc.", 2012.
- Lam C. Hadoop in action [M]. Manning Publications Co., 2010.
- Configure memory of cluster: https://hortonworks.com/blog/how-to-plan-and-configure-yarn-in-hdp-2-0/.