Exercises sheet n° 1: algorithms and programming in MapReduce and Spark

Exercise 1. Design a variant of the WordCount algorithm where the Map function performs local aggregation. The idea is to have a Map that emits couples of the kind (w, k) where k>=1 is the number of w occurrences in the input value.

Do you see any problem with this approach?

Is the combiner still needed?

Can Reduce be considered as a combiner as it is?

Observe the result of the Job, do you see any problems due to data cleaning issues?

Exercise 2. Design a MapReduce job without Combine that takes as input a file where each line contains a string indicating a Web URL and a natural number indicating the amount of time (in seconds) a user has spent on that Web page during a visit session. Of course you can have multiple lines for the same URL. The job is expected to return the list of unique URLs together with the average visit time.

Can the Reduce be considered as a Combiner?

If not, indicate the necessary modifications at the Map and Reduce level in order to have a Combine, and provide a definition of this last one.

Exercise 3. Provide a Python encoding of the following algorithms dealt with in classes: WordCount+ (Exercise 1), WordCount solving data cleaning issues, and Average calculation (Exercice2).

Exercise 4. Encoding SQL queries in MapReduce. Consider the following simple relational schema containing informations about clients and orders they made.

Customer(cid, startDate, name)

Order(#cid, total)

Note that, for the sake of simplicity, we do not have any primary key for Order. Also assume that all the fields are mandatory.

Provide the Pyhton MapReduce encoding of the following SQL queries.

- SELECT name FROM Customer WHERE month(startDate)=7
- SELECT DISTINCT name FROM Customer WHERE month(startDate)=7
- SELECT O.cid, SUM(total), COUNT(DISTINCT total) FROM Order O GROUP BY O.cid
- SELECT C.cid, O.total FROM Customer C, Order O WHERE C.cid=O.ci

For testing the jobs use the following files (use `wget` *file-url* for downloading them)

https://www.dropbox.com/s/tmt6u80mkrwfjkv/Customer.txt

https://www.dropbox.com/s/8n5cbmufqhzs4r3/Order.txt

Exercice 5. Graph analytics. Design and implement algorithms in Spark for the following two analytics problems on directed graphs. These problems are at the core of several analytics problems on graphs. Assume that a graph is a set of edge pairs (v,w) indicating that an edge exists from node v to node w.

1. **Universal sinks.** Given a directed graph, find the set of nodes having an incoming edge from all the remaining nodes, and having no outgoing edges.
2. **Triangles enumerations.** Given a directed graph, enumerate all directed triangles, with no duplicates.

These two problems are fundamental problems in many application tasks related to graph analysis, and find applications to Internet and social network analysis, just to mention a few.

Pay attention to the fact that each of these problems may require at least two jobs in order to be solved. The second job takes as input the output of the first job.

For both exercices, in order to test your Spark implementation first manually build very simple input graphs in the form of text files where each line models an edge (v, w) and contains a string for v followed by a \t , in turn followed by a string for w.

Then consider the text file graph.txt in

https://www.dropbox.com/s/3bre2e1jbsysxej/graph.txt

containing a bigger graph. Concerning the **Universal sink** exercice, in the case where there is no universal sink, find the nodes n having maximum number of (unique) nodes m connected via an (m,n) edge.

Pay attention to the fact each line the graph.txt file (obtained by a variant of the graph generator GTgraph) is of the form "v l w w" where v,l,w are integer values and l is the label for the an edge from v to w; w is repeated repeated twice. Just perform pre-processing to retain only (v,w) couples.