Big Data Management

18.06.2018

— Counting your common Facebook friends using MapReduce

Karydis Athanasios Email: dsc17008@uop.gr A.M: 17008
Voulgari Eleni Email: dsc17005@uop.gr A.M: 17005

MSc in Data Science
2nd Semester
National Centre for Scientific Research ‘Demokritos’
University of Peloponnese

Overview

In this assignment, our goal is to use MapReduce to implement the functionality of two tasks, related to the Facebook social network. The tasks are (a) counting the number of friends of a Facebook user and (b) finding the common friends between all pairs of Facebook users.

The first step is to provide the pseudocode for the Map and Reduce functions for each of the aforementioned tasks, along with a running example and an execution schematic that explains the rationale behind the provided pseudocode.

Then, the implementation of this functions, in Hadoop using Pig Latin, follows, along with some test and final files to report our findings.

The implementation of the above has been done using Hadoop pseudo-distributed and Pig Latin version 0.17.0.

Last but not least in case someone wants to have full access to our code, he may clone everything from our repository

HTTPS: https://github.com/MscBigDataManagement/FacebookFriends.git

SSH: git@github.com:MscBigDataManagement/FacebookFriends.git

Execution Instructions

In this section, we are presenting all the steps needed to execute this project. It consists of the following files:

- CountFriends.pig
- CommonFriends.pig
- friendship-20-persons.txt
- friendship-500-persons.txt
- datafu-0.0.5.jar

For each one of the tasks there is the corresponding folder, where there are all the necessary files for the execution. To be more specific, if someone wants to execute the first example, he has to cd to count_friends file and open a terminal where he can type the above commands. The first refers to the toy data and the second to the real task.

pig -param input=friendship-20-persons.txt -param output=CountResult20 -x local CountFriends.pig

and

Pig -param input=friendship-500-persons.txt -param output=CountResult500 -x local CountFriends.pig

After the execution of the scripts, two new folders with the names *CountResult20* and *CountResult500* will be created inside the folder where all the other files are. Inside these folders the files

with name *part-r-00000* will contain the results of the scripts' execution, which are lines with each facebook user and the number of his friends separated by comma.

As before, the execution of the script for CommonFriends.pig and for 20 and 500 friends respectively:

pig -param input=friendship-20-persons.txt -param output=commonresult20 -x local CommonFriends.pig

and

pig -param input=friendship-500-persons.txt -param output=commonresult500 -x local CommonFriends.pig

After the execution of the scripts, two new folders with the names *commonresult20* and *commonresult500* will be created inside the folder where all the other files are. Inside these folders, the files with name *part-r-00000* will contain the results of the scripts' execution, which are lines with pairs of facebook users and their common friends separated by space.

***The scripts have already been executed and the folders with the results already exist in the main folder. To reproduce the results, please delete all the folders containing them and then run them. If these folders already exist, the system shows it as an error.*

Count Facebook friends

The first task is to calculate the number of friends each person in the social media has.

1. (a) Below you can find the pseudocode for this task, along with the running example and an execution schematic:

Pseudocode:

```
class Mapper
        method Map(line id, user_friends = tuple[f1, f2, ….])
            name = user_friends[0]
            friends = user_friends.remove(f1)
            for all friend in friends:
                emit(name, 1)

class Reducer
        method Reduce(name, counts [c1, c2, ...])
            sum=0
            for all count c in [c1, c2, ...]
                sum = sum + c
            emit(name, sum)
```

Running example:

As an example we use the first two lines of the *friendship-20-persons.txt* file, thus the lines:

- *StteSoares lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO*
- *Fay_Fay_Lovee Hadi_Ghadanfar NinaDoctor060 cassaayruubs ___PINKbulletss tweeetmm viian_gata Love_TKO*

We assume that each line will be assigned (being the input) to one Mapper. We defined the input to be the *line id* and the tuple *user_friends*, which contains the user, whose friends we want to count, along with his friends. So, the Mapper will split the tuple into the *name*, that is the user, and a list *friends*, that is the list of his friends.

Each of the Mappers will create and emmit tuples of the form (name, 1) for each friend of the user "*name*" that the above list contains, with the key being the name of the user and the value being number one (1).

The output of the Mappers will be the following:

Mapper1: *(StteSoares, 1)*
(StteSoares, 1)
(StteSoares, 1)
(StteSoares, 1)
(StteSoares, 1)
(StteSoares, 1)

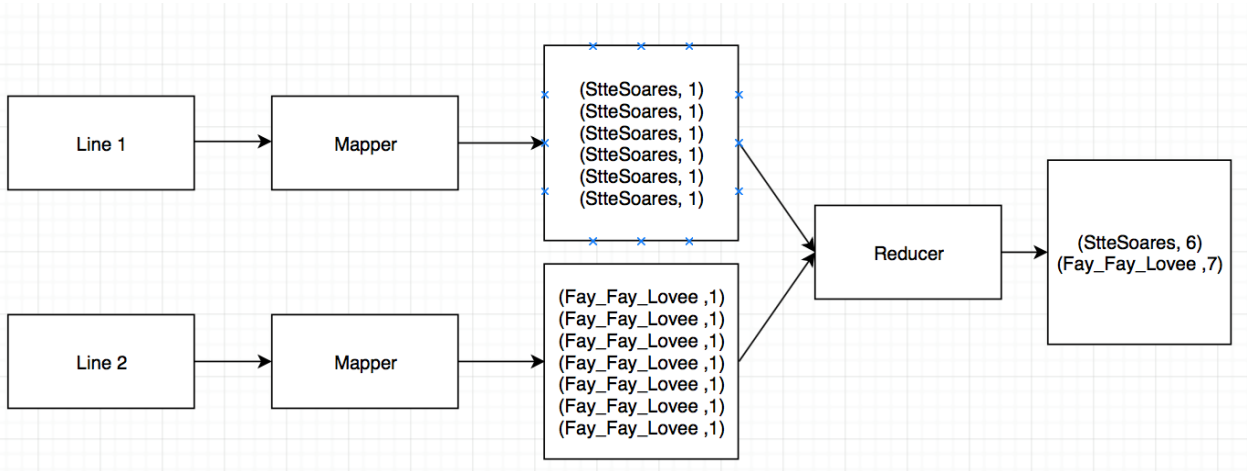
Mapper2: (Fay_Fay_Lovee ,1)
(Fay_Fay_Lovee ,1)
(Fay_Fay_Lovee ,1)
(Fay_Fay_Lovee ,1)
(Fay_Fay_Lovee ,1)
(Fay_Fay_Lovee ,1)
(Fay_Fay_Lovee ,1)

The output of the Mappers will then be merged by Job Tracker, which means they will be sorted and grouped by their key and then will be "feeded" to the Reducer, with a specific key going to the same Reducer.

The input of the Reducer will be *(StteSoares, [1,1,1,1,1,1])* and *(Fay_Fay_Lovee ,[1,1,1,1,1,1,1,1])*.

Finally, the Reducer will sum up all the numbers in the value part (list) for each key, which will lead to *(StteSoares, 6)* and *(Fay_Fay_Lovee ,7)*.

Execution Schematic:



1. (b) The comments on the implementation of the aforementioned functionality in PIG Latin can be found in the *CountFriends.pig* file. The results are the expected ones and we were able to run the complete graph, as the script successfully finds the correct number of friends of each user.

Common friends between all pairs of persons

The second task is to find the common friends between all pairs of persons in the social network.

2. (a) Below you can find the pseudocode for this task, along with the running example and an execution schematic:

Pseudocode:

```
class Mapper
        method Map(line id, user_friends = list [f1, f2, ….])
                name = user_friends[0]
                friends = user_friends.remove(f1)
                for friend in friends:
                    If name < friend:
                        emit(sorted(name, friend), friends)


class Reducer
        method Reduce(pair(name, friend), ([friends_list1], [friend_list2]))
              common_friends = []
              for all f1 in friends_list1:
                    for all f2 in friends_list2:
                          If f1 == f2:
                                common_friends.append(f1)
              emit(pair(name, friend), common_friends)
```

Running example:

As an example we use the first two lines of the *friendship-20-persons.txt* file, thus the lines:

- *StteSoares lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO*
- *mackMandaa StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro*

We assume that each line will be assigned (being the input) to a Mapper. We defined the input to be the line id and the tuple user_friends, which contains the user, whose friends we want to count, along with his friends. So, the Mapper will split the tuple into the name, that is the user, and a tuple friends, that is the list of his friends.

Each of the Mappers will create and emmit tuples of the form (sorted(name, friend), friends) with the key being the sorted pair of a user along with a friend and the value being the whole list of his friends. The pair (name, friend) is being sorted, so that all same pairs of friends will go to the same reducer.

The output of the Mappers will be the following:
Mapper1:
((lihui86, StteSoares), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO])
((Hadi_Ghadanfar, StteSoares), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO])
((cassaayruubs, StteSoares), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO])
((StteSoares, tsuyuri_ete), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO])
((mackMandaa, StteSoares), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO])
((Love_TKO, StteSoares), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO])

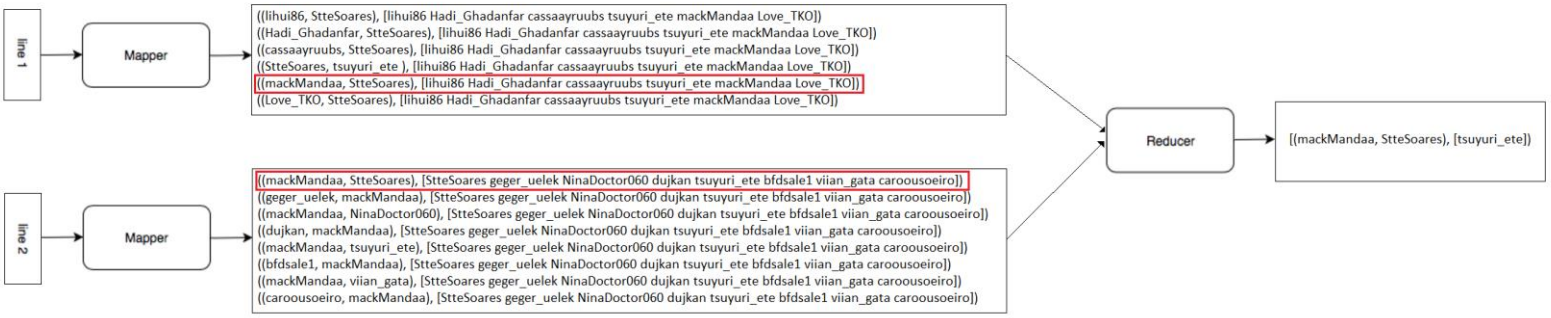
Mapper2:
((mackMandaa, StteSoares), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((geger_uelek, mackMandaa), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((mackMandaa, NinaDoctor060), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((dujkan, mackMandaa), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((mackMandaa, tsuyuri_ete), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((bfdsale1, mackMandaa), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((mackMandaa, viian_gata), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])
((caroousoeiro, mackMandaa), [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])

The output of the Mappers will be merged by Job Tracker, which means they will be sorted and grouped by their key and then will be “feeded” to the Reducer.

We focus on the Reducer, that will take the key *(mackMandaa, StteSoares)*. Its input will be:
((mackMandaa, StteSoares), [lihui86 Hadi_Ghadanfar cassaayruubs tsuyuri_ete mackMandaa Love_TKO] [StteSoares geger_uelek NinaDoctor060 dujkan tsuyuri_ete bfdsale1 viian_gata caroousoeiro])

Finally, the Reducer will perform the intersection between the two lists of friends, finding the common ones. The output of this Reducer will be:
[(mackMandaa, StteSoares), [tsuyuri_ete])

Execution Schematic:



2. (b) The comments on the implementation of the aforementioned functionality in PIG Latin can be found in the *CommonFriends.pig* file. The results are the expected ones and we were able to run the complete graph, as the script successfully finds the common friends between the pairs of the users.