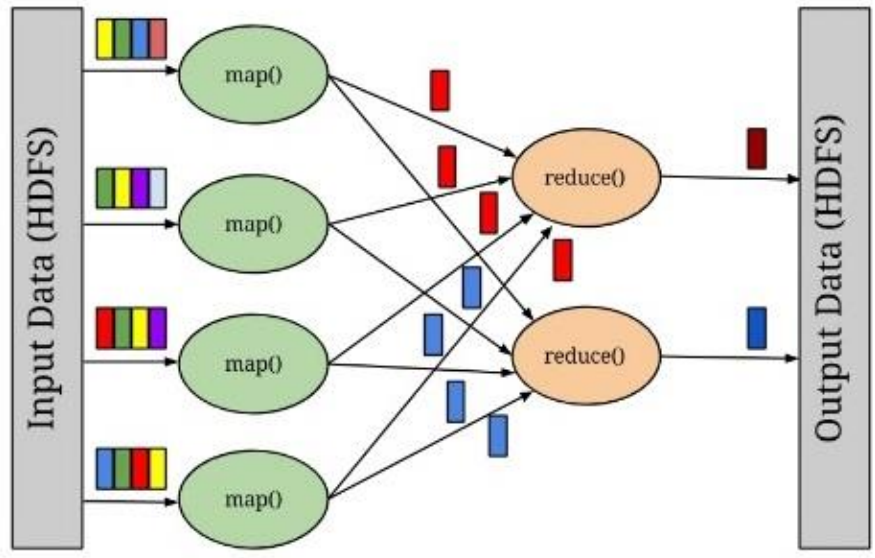
ΔΗΜΟΚΡΙΤΟΣ
ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΕΡΕΥΝΑΣ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Big Data management

2nd Exercise:

Counting your common Facebook friends using MapReduce



Alevizopoulou Sofia, dsc17002@uop.gr 2022201704002
Avgeros Giannis, dsc17003@uop.gr 2022201704003

Athens

June 2018

Contents

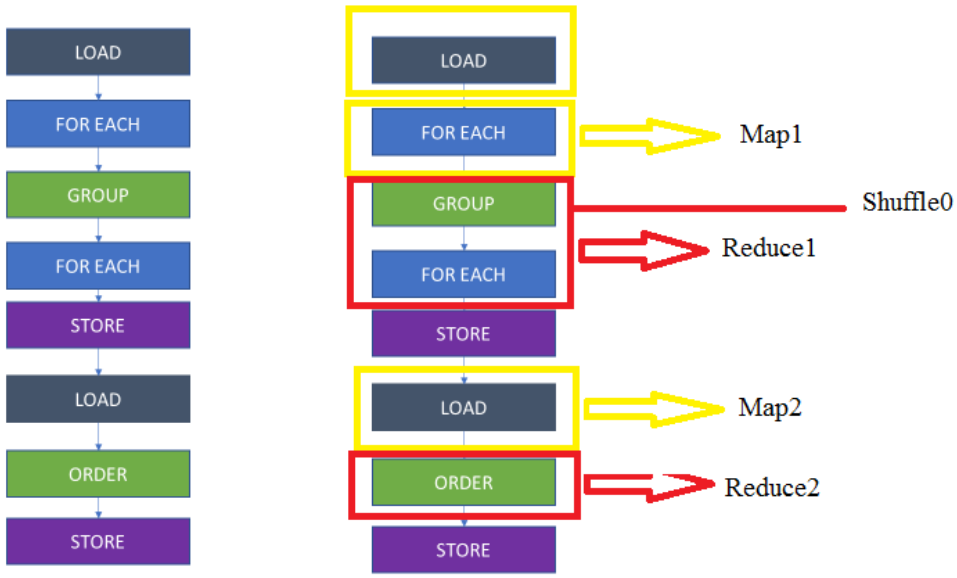
At this assignment has been implemented the functionality of finding common friends given a list of friends in facebook with MapReduce over hadoop. The first question is to calculate the number of friends that has any person on a given list and the 2nd to find the common friends between the different pairs of people.

1. Calculate the number of friends each person in the social network has

For this implementation we use Pig Latin. For the needs of the program 2 map reduces processes run. The first one is used to group the friends by name. As friendship on facebook is a bidirectional relationship; if A is a friend of B, then B is also a friend of A. So if we would like to count the friends of A it is enough to calculate how many time the person A appears in the list with friends. The second map reduce is used to sort the results from the above process.

1.1 Schematic flow

More specifically the flow that we have followed for this execution described below:



1.2 Pseudocode

The pseudocode on this diagram is described below. We have 2 MapReduce process. The 1st one is used in order to calculate the number of friends for each person and the second one in order to sort the results by value. As it is known reducer produce sorted results but only by key not by value, this functionality has to be done form another map reduce task.

The pseudocode that is described below is not 1-1 with the Pig Lattin implementation that we have implemented at the next question. Pig Lattin translates with a separate way the commands to map reduce processes and not with the obvious way. Here it is shown a pseudocode about the implementation if has done end to end by us with MapReduce.

- class Mapper1:

```
method map1 (record person, record people):
    friends=people[1…length(people)]
    for all terms t in friends do:
        Emit(t,1)
```

Input:
PersonA PersonB PersonC PersonD

We've got 'record person:A', 'record people:B, C, D'
Output:
(PersonB,1)
(PersonC,1)
(PersonD,1)

Takes as argument a line from the document, record person is the profile on facebook and record people are his friends, all together is the context of the documents line.

- class Shuffle0:

```
method shuffle0 (record all_friends):
    friendship={}
    for all terms t in record all_friends do:
            friends_of=[]
            for all terms t2 in record all_friends do:
             if t2==t:
                    friends_of.append(terms t2)
                   friendship[term t]=friends_of
      Emit(friendship)
```

Input:
(PersonA,1)
(PersonB,1)
(PersonC,1)
(PersonD,1)
(PersonA,1)
(PersonB,1)
(PersonA,1)
Outupt:
(PersonA,{1,1,1})
(PersonB,{1,1})
(PersonC,{(1})
(PersonD,{1})

Group the names by name value in order to send the whole information for one person at the same reducer. It is an intermediate result that will be used for the reducer at the next step. It is a part of the reducer1.

- *class Reducer1*:

```
method Reduce1 (record person,{1,1, ...}):
     sum=0
     for all friends f in (1, 1,,...):
            sum=sum+1
    Emit (record person, count sum)
```

Input:
(PersonA,{1,1,1})
Output:
(PersonA,3)

Each reducer will take as input the mappers' results for a specific key and make the calculations. The output file will be sortted by key. After these results from all reducer will be given as input at another MapReduce task, it will be used to sort the result by value.

- class Mapper2:

  method map2 document line):

  Emit(1,line)

Input:
(PersonA,3)
Output:
(PersonA,3)

- class sort:

  method sort (record all_friends):

  sorted=sort(people by(people.count))

  Emit(sorted)

Input:
(PersonA,1)
(PersonA,3)
(PersonB,4)
Output:
PersonB,4
PersonA,3
PersonA,1

- class reducer2:

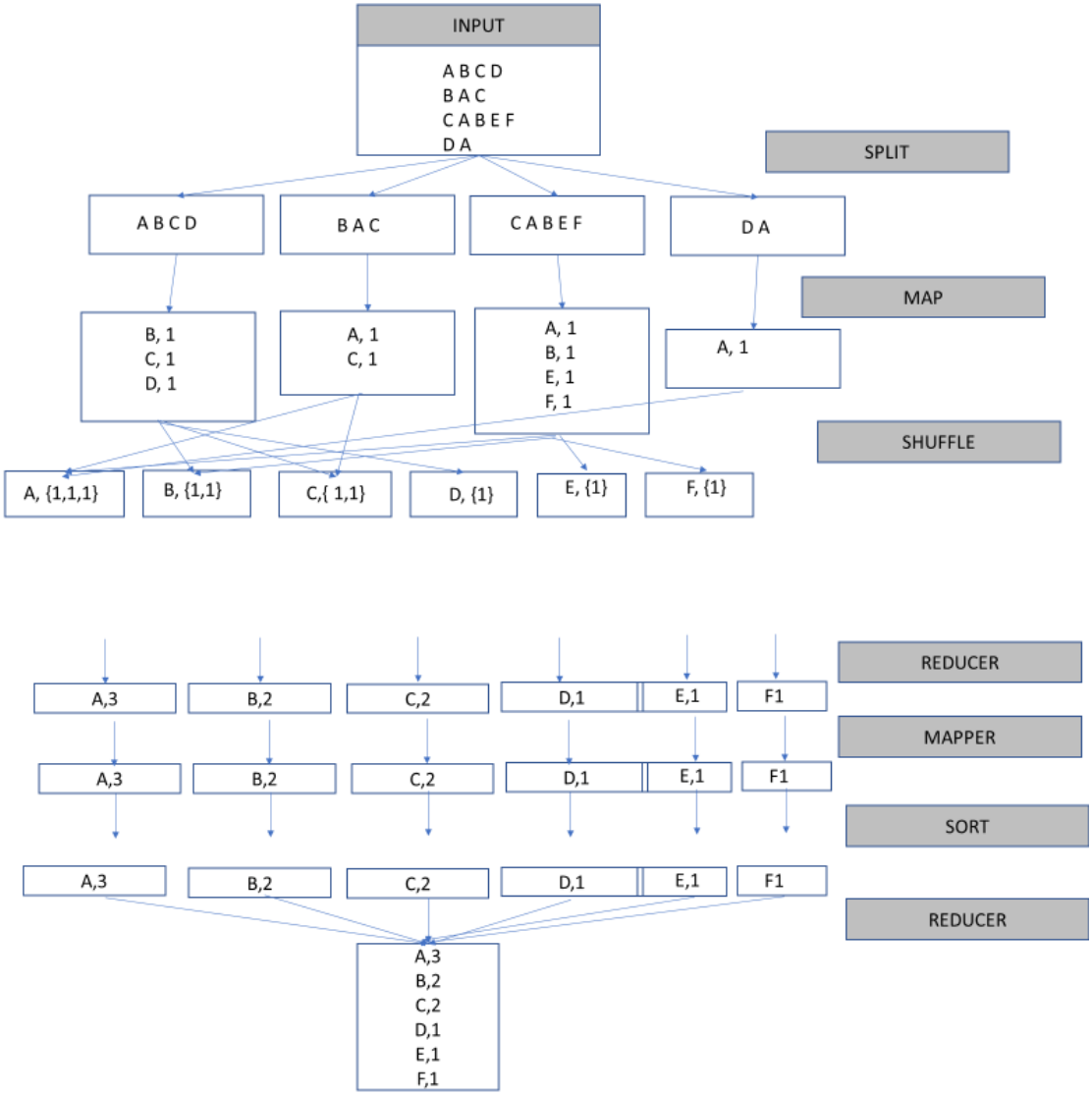
  method reduce2 (record all_friends):

  Emit(all_friends)

Input:
PersonB,4
PersonA,3
PersonA,1
Output:
PersonB,4
PersonA,3
PersonA,1

Extract the final results of the program. The results are written at a txt file.

1.3 Running example



1.3 Readme

The input of the process is a txt file contains every person in the social network followed by his/her friends. The friends list is delimited using the space character as follows:
PersonA PersonB PersonC PersonD ….

We have 2 files at which the results from the first map reduce is written. After the context of this file loads to be used as input at the 2nd map reduce.

At 'myoutput2' folder is written each person name followed by the number of its friends. At this file the results from the 1st map reduce are written. After this file is loaded and its context is sorted based on the number of friends. The final list of friends followed by the number of friends of each one is written at 'fight3' folder. At both folder the results re written at the file "part-r-00000".

Here are attached the results for the 2 inputs (50 people and 500 people).


part-r-00000-20-coun t.txt


part-r-00000-500-cou nt.txt

1.4.1 Script code

The script we run is:

```
lines = LOAD 'friendship-20-persons.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
fs -rmr myoutput2.txt
wordcount = FOREACH grouped GENERATE group, COUNT(words)-1;
STORE wordcount INTO 'myoutput2.txt' using PigStorage();
dump wordcount;
new_lines = LOAD 'myoutput2.txt/part-r-00000' AS (word:chararray,count:int);
dump new_lines;
sorted_data = ORDER new_lines BY count DESC;
fs -rmr fight3
STORE sorted_data INTO 'fight3' using PigStorage(',');
```

1.4.2 Running commands

As this part of the assignment has been implemented in Pig Lattin first you have to install Apache Pig Lattin framework.

1.4.2.1 Apache Pig Installation on Linux:

1. Step 1**:** Download **Pig tar** file.

Command: wget http://www-us.apache.org/dist/pig/pig-0.16.0/pig-0.16.0.tar.gz

2. Step 2**:** Extract the **tar** file using tar command. In below tar command, **x** means extract an archive file, **z** means filter an archive through gzip, **f** means filename of an archive file.

Command: tar -xzf pig-0.16.0.tar.gz

3. Step 3: Edit the "**.bashrc**" file to update the environment variables of Apache Pig

Command: sudo gedit .bashrc

Add the following at the end of the file:

 # Set PIG_HOME

export PIG_HOME=/home/edureka/pig-0.16.0

export PATH=$PATH:/home/edureka/pig-0.16.0/bin

export PIG_CLASSPATH=$HADOOP_CONF_DIR

4. Step 4: Run Pig to start the grunt shell. Grunt shell is used to run Pig Latin scripts.

Command: pig

Execution modes in Apache Pig:

- *MapReduce Mode* – This is the default mode, which requires access to a Hadoop cluster and HDFS installation. Since, this is a default mode, it is not necessary to specify -x flag (you can execute *pig* OR *pig -x mapreduce*). The input and output in this mode are present on HDFS.
- *Local Mode* – With access to a single machine, all files are installed and run using a local host and file system. Here the local mode is specified using '-x flag' (*pig -x local*). The input and output in this mode are present on local file system.

Command: pig -x local

5. Step 5: Execute the script

Command: run countFriends.pig

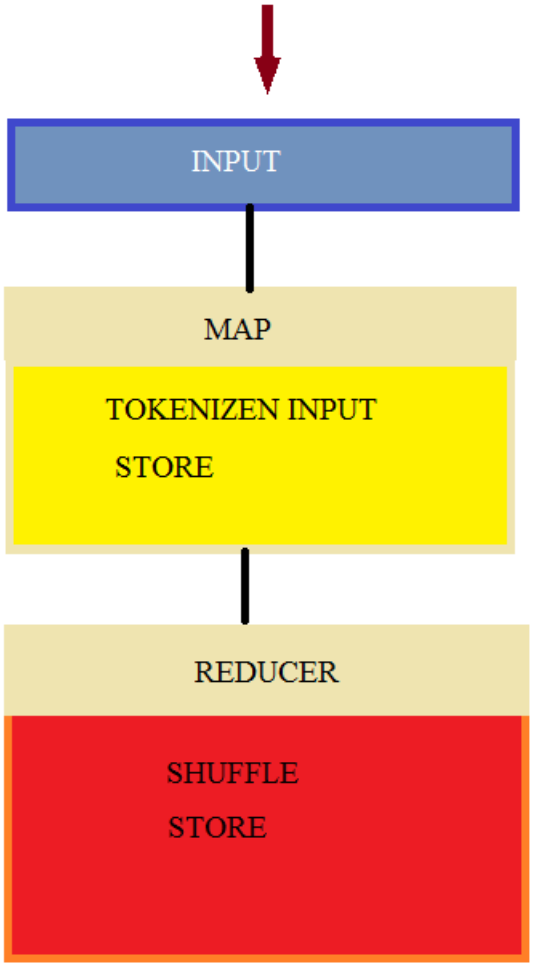
The input files have to be located at the same folder with the script.

That means that the output files are not written in HDFS.

2. Find the common friends between all pairs of persons in the social network

For this implementation we use MapReduce functionality running Hadoop. The code is written as result in Java.

2.1 Schematic flow



2.2 Pseudocode

- class mapper

```
method map( document line):
    name=line[0]
    //friends is a string item of the form Fr1,Fr2,… containing all the friends of user.
    friends=""
    for each record t in document line:
            if t.index>0:
                    friends=friends+(t)
    //Here we will create every pair consisting of user name and friend name in
    //lexicographical order
    for each record fr in friends:
            if((fr.compareTo(name))<0){
                    Emit((fr,name),friends)
            }
            else{
                    Emit((name,fr),friends)
            }
```

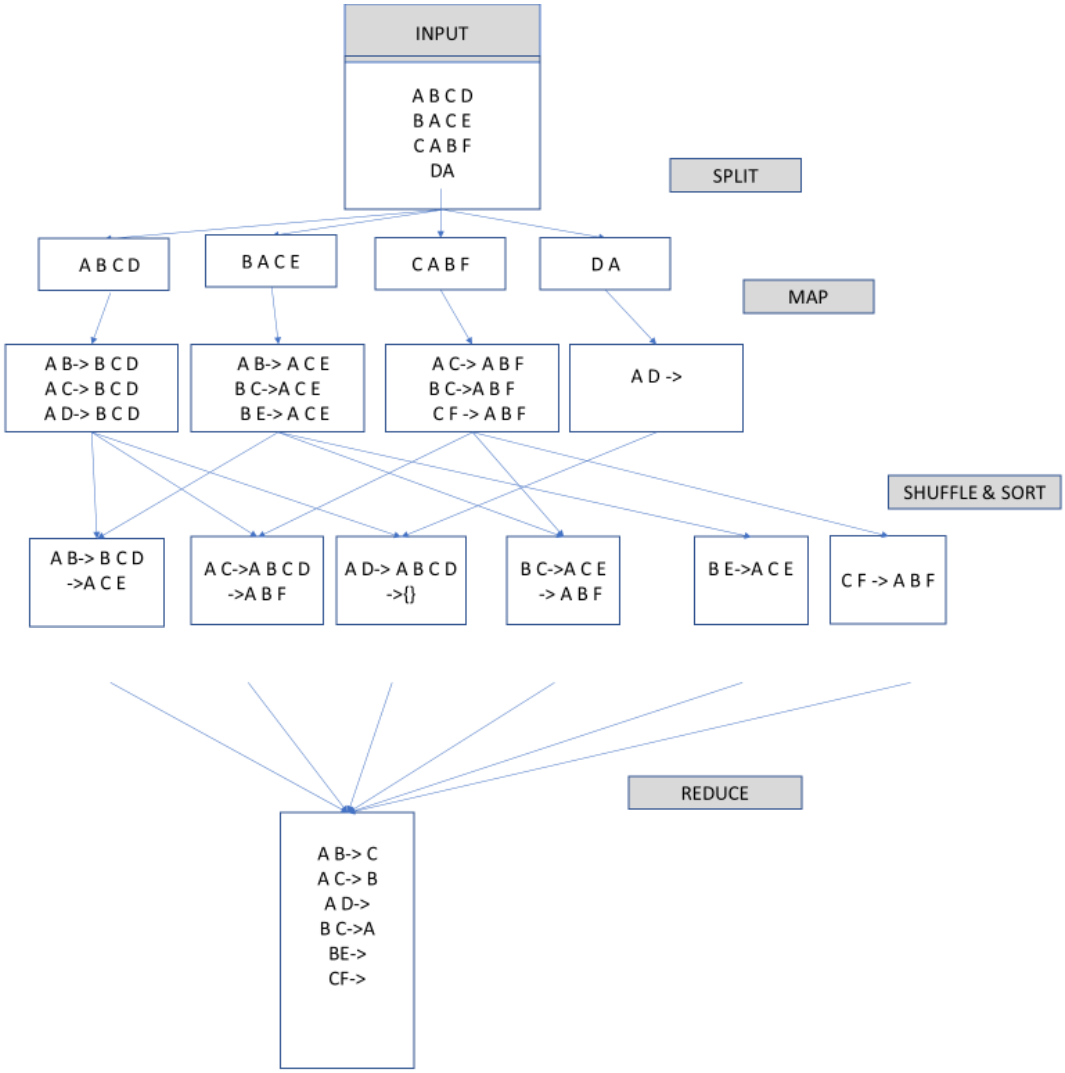
- class Shuffle (inside reducer)

```
method Shuffle( friendships f):
//pair :  (f[:0] is the first column that conatins all the pairs of friendship
 list_friends=friendships.group_by(f[:0))
 for each record t in list_friends:
    //sort friends list of a pair by the size of teh list
    sort(t.friends)
```

- class reducer

```
methos reduce( list_friends lf):
    common=""
    for each record t0 in lf.friends[0]:
            for each record t1 in lf.friends[1]:
                    it t1==t0:
                            common=common+t1
    Emit (lf.pair, common)
```

2.3 Running example



2.4 Readme

The input of the process is a txt file contains every person in the social network (e.g., PersonA followed by his/her friends. The friends list is delimited using the space character as follows:
PersonA PersonB PersonC PersonD ….
This file has been put at HDFS in order Hadoop has access.

Each mapper writes its results at a file at HDFS. Then the context of these files becomes the input for the reducer. Finally, the reducer writes its own results at another file on HDFS that has been determined from the user (running command determines the input and the output file of a Hadoop process).

As input has been given 2 files, the first is about 20 people and the second 500. The output file for 20 people is attached here, the other for 500 people is about 95MB so it isn't attached.

2.4.1 Source code

2.4.1.2 Details at the implementation that are worth noting

- Mapper:

Take as input a line from the input file and creates every pair consisting of user name and friend name in lexicographical order. For example, if the input is: B A C D then it will create-> AB, BC, BD and not BA, BC, BD. This happened in order every set of two elements (AB, BA for example) to go at the same reducer as it is about the same friendship. All these pairs followed by their friends (AB->BCD, BC->BCD, BD->BCD) will be saved at a file of HDFS. This file will be the input of the reduce process.

❖ public static class PairOfFriends extends Mapper<Object, Text, Text, Text>{

```
        public void map(Object key, Text value, Context context) throws
        IOException, InterruptedException {

                ……..

         }

}
```

- Reducer:

Inside the reducer happens the shuffle of the mappers' results. The friends of each pair at this step is a string argument, so they have to be splitted by "," and make lists with them. Next step is to combine the mappers' results grouping by the pair of friendship. After this grouping every pair will have 2 lists. The first one will be the friends from the first person of the pair and the second list will be the friends of the 2nd person of the pair. The next step is to sort descending the friendship lists of each pair. With that approach less, iterations would be required, the list with the more elements will be first. At that point the common elements between two lists for each pair should be find. A string variable will be used to store the common elements between the two users. Since the lists are sorted, a comparison between names to find if the outer element i is bigger (lexicographically) from the element j will limit the iterations. This string contains all the common friends for a pair and it will be written at the output file. The common friends are presented lexicographically.

❖ public static class FindCommonFriends extends Reducer<Text, Text,Text,Text> {

public void reduce(Text key, Iterable<Text> values,Context context) throws IOException, InterruptedException {

…….

}

}

- Main functionality:

Main function starts the jobs on mapReduce. Defines which is the mapper and reducer class and which is the input and the output file.

❖ public static void main(String[] args) throws Exception {

……

}

2.4.2 Running commands

2.4.2.1 Start the Hadoop Cluster on Linux

For starting Hadoop clustering you may follow the instruction provided at this URL http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html

After you have installed successfully the Hadoop framework the commands for running the assignment are described below. In order to be more user friendly, we have made a script at which all these commands are running.

- export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
- ssh localhost
- bin/hdfs namenode -format
- sbin/start-dfs.sh
- bin/hdfs dfs -mkdir /user
- bin/hdfs dfs -mkdir /user/mscuser
- bin/hdfs dfs -put /home/mscuser /input.txt
- bin/hadoop com.sun.tools.javac.Main /home/mscuser/ CommonFriends.java
- jar cf wc.jar CommonFriends *.class
- bin/hadoop fs -rm -r /user/mscuser/words/ log13
- bin/hadoop jar wc.jar CommonFriends /user/mscuser/input.txt /user/mscuser/log13
- bin/hadoop fs -cat /user/mscuser/log13/part-r-00000
- sbin/stop-dfs.sh

  - CommonFriend.java is the source code of MapReduce implementation.
  - Input.txt is the input file of Hadoop process
  - Log13 is the outfile of Hadoop

Both input.txt and log13 have been put into the distributed file system.

3. Attached files:

- CountFriends.pig
- CommonFriends.java and the generated classes.
- Output files except from the common friends for 500 people because of its size.