Hadoop Project: Friends Recommendation

Yupeng Gao

February 2016

1 Introduction

In this project, we want to make recommendation for the user of given ID. Our idea is that if the two friend has the most common friend but they are not mutual friends yet then We will find a list of this kind of friend. The data we use is from Stanford Large Network Dataset Collection. They are free and public online.One data set name is userdata.txt. Another one is soc-LiveJournal1Adj.txt.

We also need to do some join job using hadoop to join two table and get the information we need. The technics we use in this project are map side join, reduce side join and in memory join and job chaining.

2 Requirement One

Make recommendations for the users with following user IDs: 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993.

The key idea is that if two people have a lot of mutual friends, but they are not friends, then the system should recommend them to be connected to each other.Let's assume that the friendships are undirected: if A is a friend of B then B is also a friend of A.

The input data will contain the adjacency list and has multiple lines in the format of , where is an unique ID for an unique user, and is the list of users separated by comma who are the friends of . The following is an input example. The relationships between user and user can be understood easier in the graph.

$(user)$	$(mutual friends)$
0	$1, 2, 3$
1	$0, 2, 3, 4, 5$
2	$0, 1, 4$

In map phase, we will split the input data like (1,(2,0)), (1,(3,0)), (2,(3,0)), (3,(2,0)). if the two friends are already friends we get pair like (0,(1,-1)), (0,(2,-1)). The first number is the key. The second pair we make a class to define.

In reduce phase, the hadoop will accumulate the data according to their key. so the input of reduce side will be:

$(user)$	$list < potentialFriend, commonFriend >$
1	$(2,0),(2,1),(3,0)$
2	$(3,0),(2,0)$
3	$(2,0),(4,1)$

For each line, we can make a maintain a map(potentialFriend, list(commonFriend)).we will get:

$(user)$	$list < potentialFriend, list < commonFriend >>$
1	$(2,(3,4,5)),(4,(2,5,6))...$
2	$(4,(6,7,8)),(5,(7,8,9))...$

We can sort the list according to the size of the list by using the treeMap or sortedMap, don't forget to overwrite the compare function to make the decending order and put the potiential friend who has the most number of common friends on the top. In the end, our output format will be :

$(User)$	$list < Recommendations >$
1	$10, 23...$
2	$4, 5...$

where (User) is a unique ID corresponding to a user and (Recommendations) is a comma-separated list of unique IDs corresponding to the algorithm's recommendation of people that (User) might know, ordered by decreasing number of mutual friends. Even if a user has fewer than 10 second-degree friends, output all of them in decreasing order of the number of mutual friends. If a user has no friends, you can provide an empty list of recommendations. If there are multiple users with the same number of mutual friends, ties are broken by ordering them in a numerically ascending order of their user IDs.

3 Requirement Two

Given any two Users as input, output the list of the user id of their mutual friends by using dataset from requirement one.

For this problem, the solution is straightforward. Remember the intermediate result we got from the first requirement.

$(user)$	$list < potentialFriend, list < commonFriend >>$
1	$(2,(3,4,5)),(4,(2,5,6))...$
2	$(4,(6,7,8)),(5,(7,8,9))...$

This time we need to parse two parameter firstId, secondId to the map founction. In our main founction, we need to set the value in the configuration file.

$Configuration\ conf\ =\ new\ Configuration();$
$conf.set("firstFriend",\ args[2]);//parse\ the\ parameter\ to\ firstId$

In the map function, we need to get the parameter by write the code like:

$Configuration\ conf\ =\ context.getConfiguration();//get\ the\ parameter$
$String\ firstFriend\ =\ conf.get("firstFriend");$

After getting the parameter, what we need to do is to split the dataset and then get the corresponding list:

Output format:
$UserA, UserB \quad list < user\ id\ of\ their\ mutual\ Friends >$

4 Requirement Three

Please use in-memory join to answer this question.Given any two Users as input, output the list of the names and the zipcode of their mutual friends. Note: use the userdata.txt to get the extra user information.

Output format:
UserA id, UserB id, list of [names:zipcodes] of their mutual Friends.
Sample Output:
1234 4312 [John:75075, Jane : 75252, Ted:45045]

As we discussed in requirement two, we can get a list of mutual friends by given two user ID. But this time, we need to put two job in one program. So we need to have a job chaining at this time.

$Job1:\ \ get\ the\ list\ of\ mutual\ friends\ by\ given\ the\ two\ userId.$
$Job2:\ \ join\ the\ result\ from\ job1\ and\ userdata.txt\ and\ get\ the\ information\ we\ need.$

In our main function. we need to initial two jobs like:

$void\ main(String\ args[])\{$
$Configurationconf = newConfiguration()$
$Jobjob1 = newJob(conf, "Job1");$
$Jobjob2 = newJob(conf, "Job2");$
$\}$

For the layout in our Question3 class will be like:

$public\ class\ Question3\{$
$class\ map1\ extends\ Mapper \quad \{ \quad \}$

$class\ reduce1\ extends\ Reducer\{\quad\}//get\ the\ recommendation\ list$
$class\ map2\ extends\ Mapper\quad\{\ in\ memory\ join\}$
$class\ reduce2\ extends\ Reducer\{\quad\}\quad\}$

We need to know from this project that Hadoop provides setup and cleanup to perform preprocessing and post processing on your data. The two function will be executed only once in the map/reduce function. Usually the setup function will be used to collect some data and the cleanup function will be used to release the data. This is very important in both requirement three and requirement four.

5 Requirement Four

Using reduce-side join and job chaining: Step 1: Calculate the average age of the direct friends of each user. Step 2: Sort the users by the calculated average age from step 1 in descending order. Step 3. Output the top 20 users from step 2 with their address and the calculated average age.

Sample output.

User A, 1000 Anderson blvd, Dallas, TX, average age of direct friends.

For finding the Top K records in distributed file system like Hadoop using MapReduce we should follow the below steps:

1.In MapReduce find Top K for each mapper and send to reducer
2.Reducer will in turn find the global top 10 of all the mappers

To achieve this we can follow Top-K MapReduce design patterns which is explained below with the help of an algorithm:

$class\ mapper:$
$map(key, record):$
$insert\ record\ into\ top\ ten\ sorted\ list$
$if\ length\ of\ array\ is\ greater-than\ 10\ then$
$truncate\ list\ to\ a\ length\ of\ 10$

$cleanup():$
$for\ record\ in\ top\ sorted\ ten\ list:$
$emit\ null, record$

$class\ reducer:$
$reduce(key, records):$
$sort\ records$
$truncate\ records\ to\ top\ 10$
$for\ record\ in\ records:$
$emit\ record;$

According to Text input format we receive 1 line for each iteration of Mapper. In order to get the top 10 records from each input split we need all the records of split so that we can compare them and find the Top K records.

First step in Mapper is to extract the column based on which we would like to find Top K records and insert that value as key into TreeMap and entire row as value.

If we have more than ten records, we remove the one with the least salary as this tree map is sorted in descending order. The employee with the least salary is the last key.

Cleanup is the overridden method of Mapper class. This method will execute at the end of mapclass i.e. once per split. In clean up method of map we will get the Top K records for each split. After this we send the local top 10 of each map to reducer.

In Reducer we will get the Top 10 records from each Mapper and the steps followed in Mappers are repeated except the clean up phase because we have all the records in Reducer since key is the same for all the mappers i.e. Null-Writable.

One problem I found in this part is that I need to deal with the duplicates in the treemap collection. Because the average can be the same for different user. So treemap is not a good structure because it will overwrite the duplicates of key. What we can do is to just use a ArrayList. We maitain a list of size 20. Then each time we add a average age to the arraylist, we need to sort the list in descending order. Although it is time complexed, it can be a simple way to implement.

6 Summary

This project give me a chance to have a deep look at the hadoop as well as the real-world big data problem. We understand the structure of map/reduce and how it works and also some join technics which, I believe, is very helpful for dealing with database system and big data problem. In the future, I am looking forward to use some machine learning algorithms in hadoop to solve more complicated problems.