Friend Finding and Linking in Social Network Using Spark

Name: Manas Gaur
Affiliation: KNO.E.SIS, Wright State University, OHIO, USA
Date: November 9,2016

1 Problem Statement

Given pairs of usernames defining mutual friendships in a social network. We need to find i^{th} degree friends of each user where $i \leq N$ where N is the number of hops. We tend to make our solution scalable using parallel and distributing computing techniques.

2 Approach

In this section we will be defining an analogy of the problem which exists in the algorithm design and analysis and computability theory. Following it, we will be stating the reason of using Apache Spark over MapReduce. After that we will be stating the time complexity of the apache spark code. Lastly, we will be stating the cost incurred as a result of using Apache Spark.

2.1 Algorithmic Perspective

In our solution to the problem we have broadly used following Algorithmic Design and Analysis strategies :

1. Transitive Closure of a directed graph
2. Connected Component
3. Hash-Join and Union method

2.1.1 Cubic Solution

In this approach we tend to traverse each row of the first matrix and each column of the second matrix. Keep a note that here *N is the number of users* and *E is the number of edges.*

$$tc_{ij}^{0} = \left\{ \begin{array}{ll} 0 & \text{if } i \neq j \text{ and (i,j)} \notin E \\ 1 & \text{if i = j or (i,j)} \in E \end{array} \right\}$$

and for $N \geq 1$

$$tc_{ij}^{N} = tc_{ij}^{N-1} \vee (tc_{ik}^{N-1} \wedge tc_{kj}^{N-1})$$

In the matrix-multiplication based approach to the problem, we consider the number of users as the dimension of the adjacency matrix.In order to identify friends with i^{th} degree, we need to perform i^{th} multiplication. This approach is cubic in time and quadratic $O(N^2)$ in storage. Even on the Mapreduce framework the number of map-reduce jobs are proportional to the number of matrix multiplications and each matrix multiplication is quadratic $O(N^2)$ in time.

2.1.2 Improving Cubic Solution

The matrix multiplication involved in the problem is a boolean matrix multiplication in which Strassen's algorithm can provide improvement in time complexity using chain multiplication which is of the order of $O(N^{2.81})$ which is less then $O(N^3)$ but with the increasing size of the data (we are talking about Big Data), both the time complexity are similar. We can consider above approaches as a solution in a dense graph problems but in sparse graph problem we are losing in terms of space and time.

We require a solution with time complexity linear in size of the data (edges in the graph).

2.1.3 Hash-Join and Union Solution

This is a semi-naive algorithm for calculating the transitive closure. This algorithm is the part of *union and join* function of spark. The snippet in the code which does the union and join operation is: *tc = tc.union(tc.join(edges).mapToPair (ProjectFn.INSTANCE)).distinct().cache()*. This statement is executed over the number of users in the problem.

2.2 Apache Spark

Reason for developing the program using Apache Spark rather than Hadoop is iterative computation in which spark provide an edge over Hadoop. Resilient Distributed Datasets (RDDs) are an integral part of spark and help in processing the data on memory and it can efficiently use disk of the data does not fit into memory. Importantly, it provides fault tolerant. Properties of RDDs used in this program are

1. RDDs perform lazy computation
2. All stages in computation are synchronous
3. Built in functionality of map and filter

Since we are interested in a solution which fosters scalable and distributed computation, we tend to adapt to spark framework.

2.3 Time Complexity of the program

Spark "flatmap function" takes linear time in the order of size of the data (O(e)). Spark 'flatmaptopair function' takes linear time in the order of size of the data (O(e)). Spark "maptopair function" also takes linear time in the order of size of the data (O(e)).

Spark uses "Hash Join (O(E))" over normal Cartesian product based join (O(E^2)) which provides performance upgrades. If the union-join section is executed over a certain number of iterations (I) than hash join takes O(I*E).Also, in each iterations every user looks at its neighbours which is d neighbours and every neighbours has its d neighbours. So,we add d^2 edges for each user in each iteration. Hence, the total time complexity is $O(I(E + N * d^2))$. Since E << $N * d^2$, hence the time complexity of the program is $O(I * N * d^2)$.

2.4 Space Complexity of the Program

Spark cluster memory size should be as large as the amount of data that needs to be processed. Reason being that the data has to fit memory for optimal performance.The space complexity of the program is $O(I * N * d^2)$.Such characteristic is required for cloud processing.

2.5 Correctness of Solution

In this section we will illustrate the correctness of the solution based on the series of experiments performed with changing N value.

Table 1. Users and their friends when N=1

Users	**Friends**
brendan	torsten
torsten	brendan kim omid
mick	ziggy
ziggy	davidbowie mick
omid	davidbowie torsten
kim	davidbowie torsten
davidbowie	kim omid ziggy

Table 2. Users and their friends when N=2

Users	**Friends**
brendan	kim omid torsten
mick	davidbowie ziggy
ziggy	davidbowie kim mick omid
davidbowie	kim mick omid torsten ziggy
tor sten	brendan davidbowie kim omid
omid	brendan davidbowie kim torsten ziggy
kim	brendan davidbowie omid torsten ziggy

Table 3. Users and their friends when N=3

Users	Friends
brendan	davidbowie kim omid torsten
mick	davidbowie kim omid ziggy
torsten	brendan davidbowie kim omid ziggy
omid	brendan davidbowie kim mick torsten ziggy
ziggy	davidbowie kim mick omid torsten
davidbowie	brendan kim mick omid torsten ziggy
kim	brendan davidbowie mick omid torsten ziggy

2.6 Executing the Program

The submission of the solution will include the source code and the executable jar file. The procedure for running the jar file is

java -jar soundcloud.jar <path of the file> <number of hops>

After completion of the execution, an output folder in the name **ans_output** will be created containing following files:

1. _SUCCESS
2. part-00000
3. part-00001

These files were generated when the program was executed with the input provided as a part of this challenge. Depending on the input, the number of files generated in the **ans_output** folder will change.