Cloud Computing (INGI2145)
Assignment 1: Persons of Interest

Projetto Filippo

November 6, 2015

1 Solution description

1.1 Main idea

In order to solve the PoI problem I chose to use a parallel Search (BFS) in an iterative manner. The search starts from the source node and the neighboring nodes are visited. At the next step the neighborhood of these nodes will be visited and so on and so forth, until a PoI is found or maximum number of Hops is reached. BFS is performed by coloring the nodes and traversing according to the color of the nodes[1].

1.2 Intermediate representation

The intermediate representation I choose consists of a GraphNode java class that carries the minimal information to compute the search and some additional data to compute the set of immediate neighbors of the source node that have been used to PoIs found. In particular it stores:

- node's id of the node, as int
- node's neighbourhood, as list of Integers
- distance from source node as int
- state during the search for the node, as Color
- immediate neighbor(s) through which this node is connected to the source, as list of Integers

1.3 Implementation choices

Main implementation choices:

- *GraphNode* class implements *Writable* interface which allows Hadoop to read and write the data in a serialized form for transmission, so higher level data structures can be manipulated, without needing to use toString() to convert all your data types to text for sending over the network. Given that this data type is used in all of the MapReduce jobs, defining a specific data type classes provide a significant performance benefit[2].

[1]http://www.personal.kent.edu/ rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm
[2]https://developer.yahoo.com/hadoop/tutorial/module5.html

- custom input and output formats, it is done creating a subclass of the *FileInputFormat* and *FileOutputFormat*. This abstract class provides much of the basic handling necessary to manipulate files. Defining our custom formats allows to make easier map and reduce.
- combiners are used for optimizing bandwidth usage. During the *init* phase *GetGraphCombiner* is very useful, indeed there is an high probability that mappers will emit a lot of (key, value) pairs with the same key. It is also set a combiner for the *iteration* phase, in particular it is used the same reduce function as combiner. In this phase, the combiner seems to have less impact over the computation performance.
- *evaluate* phase, it is implemented exploiting the *setup* and *cleanup* functions of the reducer class. In the mapper, it is built a file with the adjacency list of the destination node. In the reducer, this file is read during the *setup* to build an HashSet with the adjacency list. Then, this set is used to check if a PoI was found, in this way to find the PoI for a destination node that is n hops away are needed n − 2 iterations. At the end of the reducer, in the *cleanup* function, the result is emitted.

2 Output PoIs for every test case in the file tests/orkut.txt

2.1 PoIs

- Test case 1: (srcID = 26689, dstID = 712345, maxHops = 4, numOfReducers = 1)
  Output: 2 [702788,432427,115351,46525,256199,150259,22436,62101,255144,36862,
  571025,651733,245998,33672]
- Test case 2: (srcID = 26689, dstID = 710000, maxHops = 5, numOfReducers = 1)
  Output: 2 [280037,280072,280049,709941,280086,2734245,22706,172342]
- Test case 3: (srcID = 1, dstID = 200000, maxHops = 3, numOfReducers = 1)
  Output: 1 []
- Test case 4: (srcID = 10000, dstID = 2000000, maxHops = 4, numOfReducers = 1)
  Output: 2 [49219]

2.2 Set of immediate neighbors of the source node that have been used to PoIs footnoteciao

- Test case 1: (srcID = 26689, dstID = 712345, maxHops = 4, numOfReducers = 1)
  Output: [10971,325564,2709249,76459,165874,36867,772289,49838,100722,88661,235514,
  185198,164020,221627,27414,378381,772304,233841]
- Test case 2: (srcID = 26689, dstID = 710000, maxHops = 5, numOfReducers = 1)
  Output: [772303,164960,36867,49838,100722,185386,88661,399730,235514,185198,164020,
  27414,233841]
- Test case 3: (srcID = 1, dstID = 200000, maxHops = 3, numOfReducers = 1)
  Output: []
- Test case 4: (srcID = 10000, dstID = 2000000, maxHops = 4, numOfReducers = 1)
  Output: [77658,192622,112833,96697,116555,326502,159854,126335,186041,111956,422890,21,112029,
  139120,422885,422887]

3 Elastic MapReduce cluster ID

ID: j-2INII11Q79GYV