An Engineering Challenge Not for the Faint of Heart

Having the largest portable social graph, at Rapleaf we encounter challenges everyday such as the "Super Labeler" below, a problem fundamental in figuring out the mapping of social networks to individuals. It's extremely important that the solution to a problem, like the one below, be parallelizable and highly scalable since we deal with terabytes of data.

So calling all software engineers, computer scientists, mathematicians, coders, hackers, and anyone else up to the challenge...

Think you have what it takes to top our solution to the problem below?

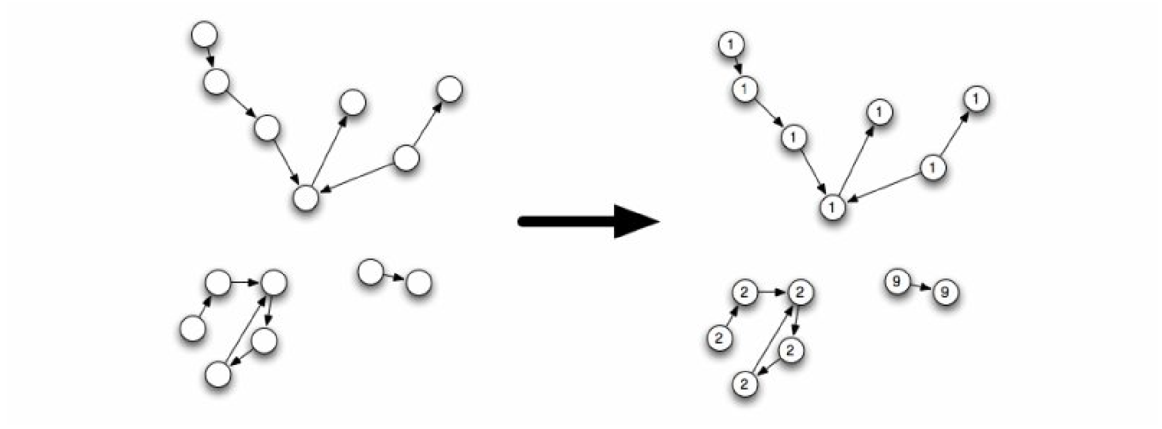
If so, send us your solution and resume at challenge@rapleaf.com. We'd love to speak to you.

Super Labeler

You run a company which specializes in labeling "anything fast, cheap, and efficient". You normally specialize in boxes, consumer items, and the like, but you were recently approached by a client with the following labeling needs:

"I have an abstract graph with billions of edges that needs to be labeled ASAP. I need each connected component to be labeled with a unique identifier. I don't care what ids you use, but the output needs to be mappings from nodes to their assigned id."

You object that you don't have the computing resources to take on such a daring task, but the client tells you that you can use his cluster of computers which has a "MapReduce" implementation installed on it. His MapReduce implementation reads and writes into a distributed filesystem which you also have access to.



As input, the client will give you a list of edges between nodes. Nodes are identified with a unique number that has been randomly chosen. There are no single-node components. As output, you need to output a mapping between nodes and unique ids for the components. Devise an efficient MapReduce algorithm to accomplish this task.

Example:

Input:
(node, node)
(5,3)
(5,7)
(1,5)
(3,2)
(9,0)

Output:
(node, component label)
(7,20)
(5,20)
(3,20)
(2,20)
(1,20)
(9,55)
(0,55)

Hint:

Hint #1: Solving this problem will require more than one MapReduce job

Hint #2: See the non-performant solution below

Be sure to us your solution and resume at challenge@rapleaf.com.

http://business.rapleaf.com/rapleaf_challenge.html

(Sub-optimal) Solution to Super Labeler Problem

The following is an example solution to the "Super Labeler" problem above. It is a subpar, non-performant solution.

The idea behind this algorithm is to produce an edge between all pairs of nodes within a component. This is accomplished by resolving one step of transitivity at a time, and continuing to do so until we reach a fixed point. For example, if our input set contains (1,2), (2,3), and (3, 4), the first iteration we will produce the new edges (1,3) and (2,4), while in the next iteration we will produce (1,4). We have reached a fixed point when an iteration does not produce any new edges.

Once that is done, we do one more pass over the data to assign a label to each node. The label chosen is the lowest identifier of the nodes in a component.

Job 1:

Make all edges bi-directional.

Map: For each tuple (a, b), emit two tuples:

(a,b)
(b,a)

Reduce: no reduce

The next step of the algorithm is to iteratively produce edges between all pairs of nodes in a component. This is accomplished by running Jobs 2 to 3b over and over until no new edges are detected.

Job 2:

Resolve transitivity and mark generated edges with a "1". Mark existing edges with a "0". For the first run, use the output of Job 1. Otherwise, use the output of last run of Job 3b.

Do one step of "transitive elimination":

Since we can only look at information "local" to a node, edges which have that node as a source, we cannot tell if the targets have edges to one another. What we will do is emit a third value with each emitted edge indicating whether that edge already exists or was generated. Afterwards, we can determine if an edge is new via the logic "an edge is new if and only if it does not have a flag marking it as 'existing'".

Map: For each tuple (a, b) emit key-value pair:

key: a
value: b

Reduce: For each key K, value list L:
For all pairs (a,b) in value list L, emit two triplets:

(a,b,1)
(b,a,1)
For key K and each value V in L, emit:

(K,V,0)

Job 3a:

Detect new edges. Uses Output of Job 2.

Map: for each 3-tuple (a,b,f), emit key-value pair:

key: (a,b)
value: f

Reduce: For key K, value list L:

If value list contains a "0", don't emit anything. Otherwise, emit (a,b)

Job 3b:

Uniqued output of iteration. Uses Output of Job 2.

Map: for each 3-tuple (a,b,f), emit key-value-pair:

key: (a,b)
value: null

Reduce: For key K, value list L

K is tuple (a,b). emit (a,b)

Now, use the distributed filesystem to see if Job 3a emitted any output. Continue repeating jobs 2 to 3b until no output is emitted by 3a which means a fixed point is reached.

Job 4:

Label each node with the smallest node identifier in a component. Uses output of Job 3b of last iteration.

Map: For each tuple (a,b):

emit key: a
emit value: b

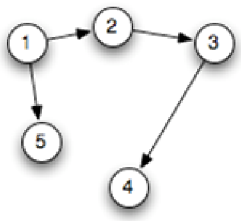
Reduce: for each key K and value list L:

Let M be the minimum value among K and all the values in L.
emit (K, M)

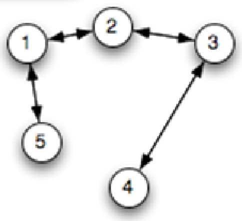
This algorithm is correct but not performant. Intuitively, the reason for the inefficiency is the need to emit all edges between nodes in a component. So the size of the set of edges will grow to around the square of the average component size. Rapleaf's algorithm, on the other hand, uses about the same number of iterations as this algorithm but does not increase the number of edges in the set in an iteration.

Illustration:

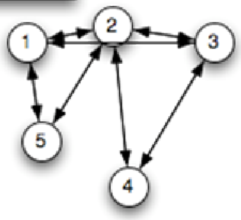
Input



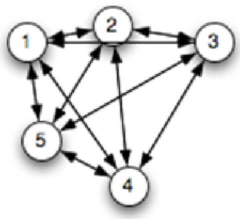
Job 1



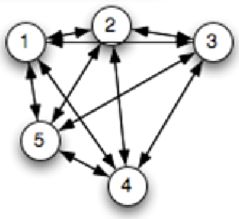
Iteration 1



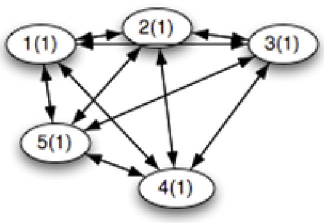
Iteration 2



Iteration 3 (no change)



Job 4 - labels in parentheses



http://business.rapleaf.com/rapleaf_challenge.html