Assignment 3

Xiaoping Li

April 13, 2014

1 Describe your Map/Reduce Algorithm

a.

Since we are required to finish counting in a single MR job. I have reduced one MR job compared with the triangle counting example provided in the assignment. Since abstract explanation will not be very clear to understand the algorithm, I will explain my algorithm by using the example from friend.simple data.

In my algorithm, mapper deals with a line of integers tokenized by whitespace. The first integer is the ID who has relation with other integer ID in the line. and generate a bunch of <key,value> pairs, where the key is a nullwritable object. The key represents an "threesome" relationship, they may not form a triangle relationship. The value could is NullWritable Object. When we build the "threesome" relationships, we will perform a sort algorithm on it.

For instance, in friendsimple data, subject 100's friend list is given as (100 200 217 300 400). The mapper will generate the following <key, value> pairs.

Table 1: Map result compare

key	value	key	value
$< 100, 200, 217 >$	null	$< 100, 117, 200 >$	null
$< 100, 200, 300 >$	null	$< 100, 200, 300 >$	null
$< 100, 200, 400 >$	null	$< 100, 200, 400 >$	null
$< 100, 217, 300 >$	null	$< 117, 200, 300 >$	null
$< 100, 217, 400 >$	null	$< 117, 200, 400 >$	null
$< 100, 300, 400 >$	null	$< 200, 300, 400 >$	null

The intemediate results from mappers are gathered and sorted/shuffled by key. The sorted intermediate results are then sent to reducers as inputs.

The reducer takes the intermediate results for analysis. Since the intermediate results has been sorted, we can guarantee if there are two or more same keys, this key forms a triangle relation.

For example, in the above intermediate results, key<100,200,300> key <100,200,300> , so <100,200,300> is a triangle relation. Since 100 has relation with 200 and 300, 200 also has relation with 100 and 300. Thus it forms a triangle relationship.

Table 2: Reducer Result For Key <100,200,300>

<100,200,300>	null
<100,200,300>	null

Since we have presumed the mutable relationship between friend A and friend B which means when A is a friend of B, B is also a friend of A. This property has guarantee our algorithm works right. In above example, we can figure out ID 100, ID 200, ID 300 forms a mutual relationship. For the first key, <100,200,300> , ID

100 has relation to ID200 and ID 300. For the second key in another mapper, <100,200,300> , it actually comes from <200,100,300>. And we have counted 2 same key <100,200,300>, thus we print the user ID 100, 200 and 300 as a triangle relationship.

So for different mapper, if they have generate a same key, then the ID in the key forms a triangle relation. We also can prove that if the key is unique, then it is not triangle, since we have sorted all the key list, there is no triangle relation between them. For example, $< 100, 200, 217 >$ is the unique key during the example. Since if they are triangle relation, based on the presumed property, User 100 has relation with 200, so 200 must has relation with 100, if 200 has relation to 217, then there must also exist the same key $< 100, 200, 217 >$ which leads the count equal to 2.

Based on above discussion, the algorithm is robust.

b.

The potential parallelism lies in both the mapping, sorting/shuffling and reducing. For example, for friends1000 data, 1000 mappers at most is allowed for. Some tasks are pending when others are running because only 4 slave instances are available. With enough slaves, these tasks could run simultaneously in a really short time. The max number of reducers allowed varies with the graph. The max number of reducers should be no more than the number of edges in the graph.

c.

The input for the Mapper is a line of whitespace or tab separated integers representing the friends list of a subject. (Exactly speaking, it should also be a key-value pairs, the key is Long Integer assigned by Hadoop) The output for the mapper is a new key-value pair, where each key represents an a triad. The motivatin for this output is to find the same key for the mapper which means that these keys forms a triangle relationship. And since the value for the key is NullWritable object. So it is easier to just deal with keys. So it also can improve performance for the map/reduce. When sort/shuffled, the intermediate reulst can be consumed by the reducers.

2 On Combiners

a.

Combiner function is always used as an optimization for MapReduce job. Combiners is helpful in that it helps reducing the information for sent/received among slaves. However, in this application, it is not applicable, so we have to leave it empty. Combiners can only be used in specific cases which are going to be job dependent. In this application, each mapper's generated information is not enough to make a decision of triangle counting. This is because only when we have the **full value list** for a key, we can find out that whether there are same keys existing, if there are duplicate keys, print the three friend ID as a triangle relation. So only when we have all keys, then we can made next decision.

b.

We can learn from this applicaiton, combiners can't be used when next step needs additional information. Combiner is used for fucntions which are associative and commutative. For example, in word count, we can use combiners since the function is addition. Another constraint that a combiner has is that the input/output key and value types must match the output types of the Mapper. So if we input string: integer type, then our output type can't be string:float.

3 Analysis

a.

To get the total number of triangle relations, each node has a fixed degree l, the time complexity is $O(l^2n)$ which is based on the sacraficed space complexity: $O(n^2)$ space.(Good time complexity is built based on bad space complexity) For each person, we should iterate its triad which occupies $O(l^2)$ time complexity. Since there are n different person, the running time for the serial implementation should be $O(l^2n)$ according to above analysis.

b.

The Mapper generated $O(nl^2)$ of intermediate results and since we have performed sorting algorithm which cost $O(nl^2 \log(nl^2))$ time complexity. The reducer takes $O(nl^2)$ of time complexity. So the overall time complexity is $O(nl^2 \log(nl^2)) = O(nl^2 log(n))$.

c.

Suppose we have M mappers, since the total work of Map is $O(nl^2)$, thus for each mapper work should be $O(\frac{nl^2}{M})$. Suppose we have maximum mapper n, then each mapper only does $O(n^2)$ work. Same as mappers, if we have R reducers, then each reducer also does $O(\frac{nl^2}{R})$ of work.

d.

The above conclusion has implied that large parallelism projects should avoid complexity. We can see the comparison from the best serial implementation and map/reduce. The consequece is, to increase parallelism, we need to reduce complexity. When the number of computing units larger than some limits , then parallelism will dominate complexity. We need to consider the hardware condition, one factor to decide the number of mapper and reducer is based on how many cores do we have. If we double the worksize, $O(\frac{nl^2}{R}) + O(\frac{nl^2}{M}) = O(\frac{M+N}{MN})\ O(nl^2)$, then the operation time for parallel should be reduced.