Homework #5 – MapReduce

Yoke Kai Wen, 2020280598

March 26, 2021

1 Introduction

In this assignment, I read the MapReduce paper and implemented a MapReduce program counting the out-degree of each vertex in a graph.

2 Task 1: Paper reading

2.1 Propose a way to significantly reduce network traffic when results of mapping are shuffled and sent to reducers.

Implement locality optimisation - each worker should access data on its own local machine as much as possible to avoid sending and receiving data across the network to and from other machines. For instance. mapper machines should store only a single copy of intermediate data to its local disk (no need to store in master which would involve more network traffic), and nearby machines (e.g. on same network switch) should be assigned corresponding reducing tasks, so that the reducer machine can easily access the intermediate data on the nearby mapper machine without costing a lot of network traffic. Mapping tasks should also be assigned to machines by the master such that each mapper can read input data directly from their local disks or nearby local disks, instead of waiting for input data to be sent from far-away machines.

2.2 Propose an approach to tackle slow mappers and reducers.

Implement redundant execution: When a MapReduce operation is nearing completion, the master should schedule back-up executions of remaining in-progress tasks. Task is marked complete whenever primary or back-up execution completes, hence it does not matter if a particular machine is slow at executing a mapping/reducing task as some other machines would have completed the same task.

3 Task 2: Implementing program to count out-degree of all vertices in a graph using MapReduce

I implemented Task 2 by modifying the given `OutDegree.java` script and running the given `run_od.sh` bash script for compilation and running on server. For this task, we can ignore the weights of the edges, and count repeating edges as multiple edges. Hence, this means we only have to count the occurrences of each source node in the given edge file. This is essentially the same problem as the WordCount problem for which an example script was provided, except that the input text file format is slightly different. Hence, I copied the Map and Reduce functions from `WordCount.java` and modified the Map function slightly as shown in the code fragment below, while making no changes to the Map function.

```
    StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
          if (itr.nextToken().equals("a")) {
            word.set(itr.nextToken());
                  context.write(word, one);
          }
            }
```

3.1 Map

The input edge file takes the form of ’a u v w’, where ’a’ marks the start of a new line, u marks the source node, v the destination node, and w the weight. For Task 2, we need to find the number of occurrences of each node ’u’, which is the next word token after ”a”. Hence, the Map function checks whether a string token’s value is ”a”, and if so, then we store the next string token which would represent the source node, and assign a count of 1 to that source node string token. Thus, the intermediate key is the source node string, and the value is ’1’.

3.2 Reduce

These intermediate keys are then sent to the reducer, which accumulates the counts of each unique intermediate key, and outputs finally the occurrences of each source node in the edgelist, in the exact same way as Reduce in for WordCount. This represents the outdegree of each node in the graph.

4 Task 3: Find top-20 biggest out-degree vertices using MapReduce

I did not have time to implement the code for Task 3, but I have a general idea. The input file to the Map task would be the out-degree counts from Task 2, in the form of (key, value) – (node, out-degree). The Map task then swaps the input key and value around to form the intermediate keys and values, in the form of (out-degree, node). Since MapReduce offers ordering guarantees by ensuring that the intermediate key/value pairs are processed in increasing key order, the reducing function is merely an identity function that copies the supplied intermediate data to the output, but with swapped key and value as (node, out-degree). To obtain the top-20 biggest out-degree vertices, simply look at the last 20 lines of the output file.