Big Data HW5

Tiago Melo Antunes 2020280599

March 2021

1 Task 1

1.1 Q1

To lower the network traffic, we can perform a small grouping after sorting to ease the network traffic. We collect multiple entries in a row with the same key and group them in the same line, thus yielding a line of the type:

```
<key> <val1> <val2> ... <valn>
```

instead of multiple lines of

```
<key> <val>
```

pairs. The authors mentioned this in the paper, which they called the **combiner**. This works as a kind of reducer that acts in between the map finishes and writes to the intermediate map output.

1.2 Q2

Both mappers and reduces, if they are slow, will make the overall task very slow since they need to synchronize between the map and reduce phase, and after the reduce phase. A way to avoid having a single task performing too slow is for the master to give the same task to another worker after some time. The result that finishes first is the one that is considered as finishing the job. This allows the system to recover from slow nodes impacting the performance. This approach works both in maps and reduces. These are called **backup tasks** in the paper.

2 Part 2

My solution is quite simple. Map functions just output

```
<start node> 1
```

and the reduce function sums these values and outputs

```
<start node> <sum>
```

which results in the out degree of each node, while also counting the repeat edges as requested.

To run my solution, just run the given script *run_od.sh*.

3 Part 3

My solution uses 2 MapReduces to perform the calculation in sequence. The first MapReduce is the same as in Part 2. The second one swaps the keys and the values so we have

```
<out degree> <node>
```

I then gave it a customized sorting function which will sort in decreasing order. This will sort the keys in descending order, giving the nodes with highest out degree first. Then I also gave a parameter to the reduce function that specifies the number of elements I wanted to see being output.

Because we're only having a one file example, I did not worry much with this, but in case we had multiple files, this would not work. It would require a third map reduce in which the map function (only 1 map worker) simply outputs the input, and the reduce function then performs the same operation over the input. The second MapReduce however would be outputting the top k elements for each map function, and the third one would be outputting the top k of each top k. In my implementation, the reduce function outputs

```
<out degree> <node>
```

but limits the amount of times it prints to be k.

To run this optional solution, run the *./run_od_opt.sh* instead. Running it will give the answer to both Part 2 and 3.