

# CS4225/CS5425 BIG DATA SYSTEMS FOR DATA SCIENCE

## Tutorial 1: MapReduce

1. Suppose our input data to a map-reduce operation consists of integer. The map function takes an integer  $i$  and produces the list of pairs  $(p, i)$  such that  $p$  is a prime divisor of  $i$ . For example,  $\text{map}(12) = [(2,12), (3,12)]$ .

The reduce function is addition. That is,  $\text{reduce}(p, [i_1, i_2, \dots, i_k])$  is  $(p, i_1 + i_2 + \dots + i_k)$ .

Given that the input is the list  $[15, 21, 24, 30, 49]$ , identify all the pairs in the output.

2. We have a large matrix ( $n \times m$ , and  $n \gg m$ ) that cannot be stored on a single machine. So we distribute it on a cluster of machines. We want to compute the result of matrix-vector multiplication. Since  $n \gg m$ , the vector is stored on every machine of the cluster. The input to the mapper function is (value, row, col). And vector  $v$  is already known to each mapper.

1	2	3	4	1
5	6	7	8	2
9	10	11	12	3
13	14	15	16	4

Design a suitable map and reduce function, and identify all the pairs in the output.

(Note: your Map function only needs to take the matrix elements as input; the vector can be treated as fixed. Hint: For each matrix element, think about what computation needs to be done on it for computing matrix multiplication (for the “Map”). Then think about which of the resulting outputs need to be aggregated, to help you decide what the intermediate keys should be.)

3. Consider a simple example: we have a large dataset where input keys are strings and input values are integers, and we wish to compute the mean of all integers associated with the same key (rounded down). A real-world example might be a large user log from a popular website, where keys represent user ids and values represent some measure of activity such as elapsed time for a particular session. A program Tommy has implemented the problem on MapReduce. He has written a few versions with the pseudo code shown in Figures 1—3.

a) Initially, Tommy has finished an implementation with Version 1 (Figure 1). He finds that the implementation can have correct results, but the performance is very slow. Why?

b) Tommy wants to improve the performance using combiner. He comes out the second implementation (Version 2 in Figure 2). He finds that he cannot run the code successfully. Why?

c) After careful design, Tommy finally develops an efficient and correct implementation (Version 3 in Figure 3). Analyze the correctness of the combiner and efficiency of the algorithm (i.e., why it is more efficient than Version 1).

```

1: class MAPPER
2:   method MAP(string  $t$ , integer  $r$ )
3:     EMIT(string  $t$ , integer  $r$ )
1: class REDUCER
2:   method REDUCE(string  $t$ , integers  $[r_1, r_2, \dots]$ )
3:      $sum \leftarrow 0$ 
4:      $cnt \leftarrow 0$ 
5:     for all integer  $r \in$  integers  $[r_1, r_2, \dots]$  do
6:        $sum \leftarrow sum + r$ 
7:        $cnt \leftarrow cnt + 1$ 
8:      $r_{avg} \leftarrow sum / cnt$ 
9:     EMIT(string  $t$ , integer  $r_{avg}$ )

```

*Figure 1. Computing the average: Version 1*

```

1: class MAPPER
2:   method MAP(string  $t$ , integer  $r$ )
3:     EMIT(string  $t$ , integer  $r$ )
1: class COMBINER
2:   method COMBINE(string  $t$ , integers  $[r_1, r_2, \dots]$ )
3:      $sum \leftarrow 0$ 
4:      $cnt \leftarrow 0$ 
5:     for all integer  $r \in$  integers  $[r_1, r_2, \dots]$  do
6:        $sum \leftarrow sum + r$ 
7:        $cnt \leftarrow cnt + 1$ 
8:     EMIT(string  $t$ , pair  $(sum, cnt)$ ) ▷ Separate sum and count
1: class REDUCER
2:   method REDUCE(string  $t$ , pairs  $[(s_1, c_1), (s_2, c_2) \dots]$ )
3:      $sum \leftarrow 0$ 
4:      $cnt \leftarrow 0$ 
5:     for all pair  $(s, c) \in$  pairs  $[(s_1, c_1), (s_2, c_2) \dots]$  do
6:        $sum \leftarrow sum + s$ 
7:        $cnt \leftarrow cnt + c$ 
8:      $r_{avg} \leftarrow sum / cnt$ 
9:     EMIT(string  $t$ , integer  $r_{avg}$ )

```

*Figure 2. Computing the average: Version 2*

```

1: class MAPPER
2:   method MAP(string  $t$ , integer  $r$ )
3:     EMIT(string  $t$ , pair ( $r$ , 1))
1: class COMBINER
2:   method COMBINE(string  $t$ , pairs  $[(s_1, c_1), (s_2, c_2) \dots]$ )
3:      $sum \leftarrow 0$ 
4:      $cnt \leftarrow 0$ 
5:     for all pair  $(s, c) \in$  pairs  $[(s_1, c_1), (s_2, c_2) \dots]$  do
6:        $sum \leftarrow sum + s$ 
7:        $cnt \leftarrow cnt + c$ 
8:     EMIT(string  $t$ , pair ( $sum$ ,  $cnt$ ))
1: class REDUCER
2:   method REDUCE(string  $t$ , pairs  $[(s_1, c_1), (s_2, c_2) \dots]$ )
3:      $sum \leftarrow 0$ 
4:      $cnt \leftarrow 0$ 
5:     for all pair  $(s, c) \in$  pairs  $[(s_1, c_1), (s_2, c_2) \dots]$  do
6:        $sum \leftarrow sum + s$ 
7:        $cnt \leftarrow cnt + c$ 
8:      $r_{avg} \leftarrow sum / cnt$ 
9:     EMIT(string  $t$ , pair ( $r_{avg}$ ,  $cnt$ ))

```

*Figure 3. Computing the average: Version 3*