CS4225/CS5425 BIG DATA SYSTEMS FOR DATA SCIENCE

Tutorial 3: MapReduce

1. Suppose our input data to a map-reduce operation consists of integer values (the keys are not important). 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, map(12) = [(2,12), (3,12)].

The reduce function is addition. That is, reduce(*p*, [i_1, i_2, ...,i_k]) is (p,i_1+i_2+...+i_k).

Compute the output, if the input is the set of integers 15, 21, 24, 30, 49. Then, identify all the pairs in the output.

2. Using the matrix-vector multiplication described in Section 2.3.1, applied to the matrix and vector:

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

apply the Map function to this matrix and vector. Then, identify all the key-value pairs that are output of Map.

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 average value of all integers associated with the same key (rounded to the nearest integer). 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 developer Tommy has implemented the problem on MapReduce. He has written a few versions with the pseudo code shown in Figures 1—4.

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 poor. 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 can seldom get the correct results. 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).

d) Tommy analyzes the efficiency of Version 3, and comes out an even more efficient implementation (Version 4 in Figure 4). Why does Version 4 is even more efficient than Version 3?

```
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, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← 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, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← 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) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← 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) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← 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) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, pair (r_avg, cnt))
```

Figure 3. Computing the average: Version 3

```
1: class Mapper
2:     method Initialize
3:         S ← new AssociativeArray
4:         C ← new AssociativeArray
5:     method Map(string t, integer r)
6:         S{t} ← S{t} + r
7:         C{t} ← C{t} + 1
8:     method Close
9:         for all term t ∈ S do
10:            Emit(term t, pair (S{t}, C{t}))
```

Figure 4. Computing the average: Version 4