CS4225/CS5425 BIG DATA SYSTEMS FOR DATA SCIENCE

Tutorial 1: 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).

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

Answer:

- ***Map does the following:***
  - 15 -> (3,15), (5,15)
  - 21 -> (3,21), (7,21)
  - 24 -> (2,24), (3,24)
  - 30 -> (2,30), (3,30), (5,30)
  - 49 -> (7,49)
- *Then group by keys:*
  - (2, [24,30])
  - (3, [15,21,24,30])
  - (5, [15,30])
  - (7, [21,49])
- *Reduce add elements:*
  - (2,54)
  - (3,90)
  - (5,45)
  - (7,70)

2. We want to use map-reduce to compute the result of matrix-vector multiplication of the following matrix and vector:

$$\begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \quad \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 3 \\ \hline 4 \\ \hline \end{array}$$

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.)

Answer:

- Each m_{ij} is multiplied by v_j, and this product forms the value of a key-value pair that has key i, the row number.
- Thus, in row-major order, the sixteen key-value pairs produced are:

(1,1)	(1,4)	(1,9)	(1,16)
(2,5)	(2,12)	(2,21)	(2,32)
(3,9)	(3,20)	(3,33)	(3,48)
(4,13)	(4,28)	(4,45)	(4,64)

- Reduce will sum the values corresponding to the same key, thus, results of reduce are:
- (1,30), (2,70), (3,110), (4,150)

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—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 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 can seldom get the reasonable 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 is Version 4 even more efficient than Version 3?

Answer:

a) It requires shuffling all key-value pairs from mappers to reducers across the network, which is highly inefficient.

b) Recall that combiners must have the same input and output key-value type, which also must be the same as the mapper output type and the reducer input type. This is clearly not the case. To understand why this restriction is necessary in the programming model, remember that combiners are optimizations that cannot change the correctness of the algorithm. So let us remove the combiner and see what happens: the output value type of the mapper is integer, so the reducer expects to receive a list of integers as values. But the reducer actually expects a list of pairs! The correctness of the algorithm is contingent on the combiner running on the output of the mappers, and more specifically, that the combiner is run exactly once.

c) In the mapper we emit as the value a pair consisting of the integer and one—this corresponds to a partial count over one instance. The combiner separately aggregates the partial sums and the partial counts (as before), and emits pairs with updated sums and counts. The reducer is similar to the combiner, except that the mean is computed at the end. In essence, this algorithm transforms a non-associative operation (mean of numbers) into an associative operation (element-wise sum of a pair of numbers, with an additional division at the very end).

d) Inside the mapper, the partial sums and counts associated with each string are held in memory across input key-value pairs. Intermediate key-value pairs are emitted only after the entire input split has been processed; similar to before, the value is a pair consisting of the sum and count.

```
class Mapper
    method Map(string $t$, integer $r$)
        Emit(string $t$, integer $r$)

class Reducer
    method Reduce(string $t$, integers $[r_1, r_2, \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all integer $r \in$ integers $[r_1, r_2, \ldots]$ do
            $sum \leftarrow sum + r$
            $cnt \leftarrow cnt + 1$
        $r_{avg} \leftarrow sum/cnt$
        Emit(string $t$, integer $r_{avg}$)
```

Figure 1. Computing the Mean: Version 1

```
class Mapper
    method Map(string $t$, integer $r$)
        Emit(string $t$, integer $r$)
class Combiner
    method Combine(string $t$, integers $[r_1, r_2, \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all integer $r \in$ integers $[r_1, r_2, \ldots]$ do
            $sum \leftarrow sum + r$
            $cnt \leftarrow cnt + 1$
        Emit(string $t$, pair $(sum, cnt)$)                ▷ Separate sum and count
class Reducer
    method Reduce(string $t$, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ do
            $sum \leftarrow sum + s$
            $cnt \leftarrow cnt + c$
        $r_{avg} \leftarrow sum/cnt$
        Emit(string $t$, integer $r_{avg}$)
```

Figure 2. Computing the Mean: Version 2

```
class Mapper
    method Map(string $t$, integer $r$)
        Emit(string $t$, pair $(r, 1)$)
class Combiner
    method Combine(string $t$, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ do
            $sum \leftarrow sum + s$
            $cnt \leftarrow cnt + c$
        Emit(string $t$, pair $(sum, cnt)$)
class Reducer
    method Reduce(string $t$, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ do
            $sum \leftarrow sum + s$
            $cnt \leftarrow cnt + c$
        $r_{avg} \leftarrow sum/cnt$
        Emit(string $t$, pair $(r_{avg}, cnt)$)
```

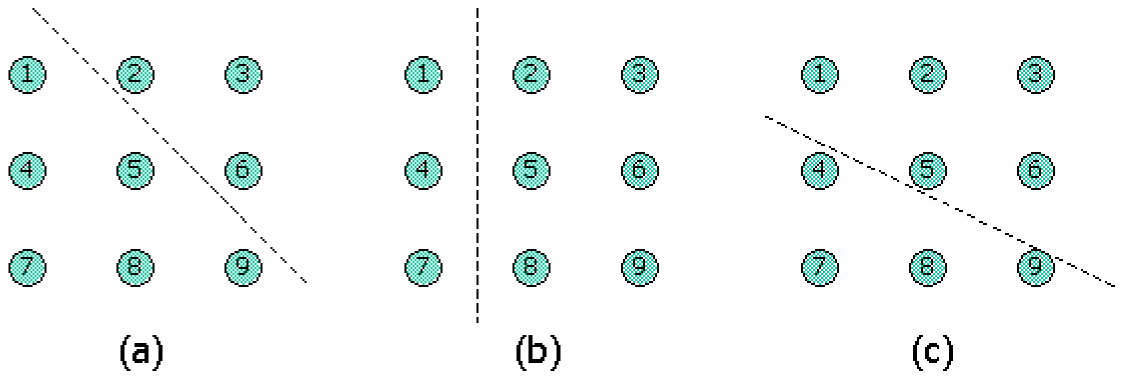
Figure 3. Computing the Mean: Version 3

```
class Mapper
    method Initialize
        $S \leftarrow$ new AssociativeArray
        $C \leftarrow$ new AssociativeArray
    method Map(string $t$, integer $r$)
        $S\{t\} \leftarrow S\{t\} + r$
        $C\{t\} \leftarrow C\{t\} + 1$
    method Close
        for all term $t \in S$ do
            Emit(term $t$, pair $(S\{t\}, C\{t\})$)
```

Figure 4. Computing the Mean: Version 4

4. (Additional Question, Optional) The Bisecting k-Means algorithm starts by dividing the points into two clusters. It may consider several bisections and pick the best one. Let us take "best" to mean the lowest SSE (Sum Squared Error). The SSE is defined to be the sum of the squares of the distances between each of the points of the cluster and the centroid of the cluster.

Suppose that the data set consists of nine points arranged in a square grid, as suggested by the figure below:



Although it doesn't matter for this question, you may take the grid spacing to be 1 (i.e., the squares are 2-by-2) and the lower-left corner to be the point (0,0). We see in the figure three possible bisections. (a) would be the bisection if we chose the two initial centroids to be 3 and 7, for example, and broke ties in favor of 7. (b) would be the split if we chose initial centroids 1 and 2. (c) would be the split for initial choice 2 and 7. Rank these three options from the best to the worse choice.

Answer:

For (a), the centroids of the two clusters are (2/3, 2/3) and (5/3, 5/3). Since the cluster of the first consists of the points (0,0), (0,1), (0,2), (1,0), (1,1), and (2,0), the SSE is 20/3. The other cluster, consisting of the points (1,2), (2,1), and (2,2), has SSE 4/3, so the total for this bisection is 8.

For (b), the centroids are (0,1) and (1.5,1), and the SSE's for these clusters are 2 and 5.5, respectively. Thus, this bisection has an SSE of 7.5.

For (c), the centroids are (0.75,0.25) and (1.2,1.6). The SSE's for these clusters are 3.5 and 4, respectively. Thus, this bisection also has an SSE of 7.5. We conclude that (b) and (c) are equally good, and better than (a).