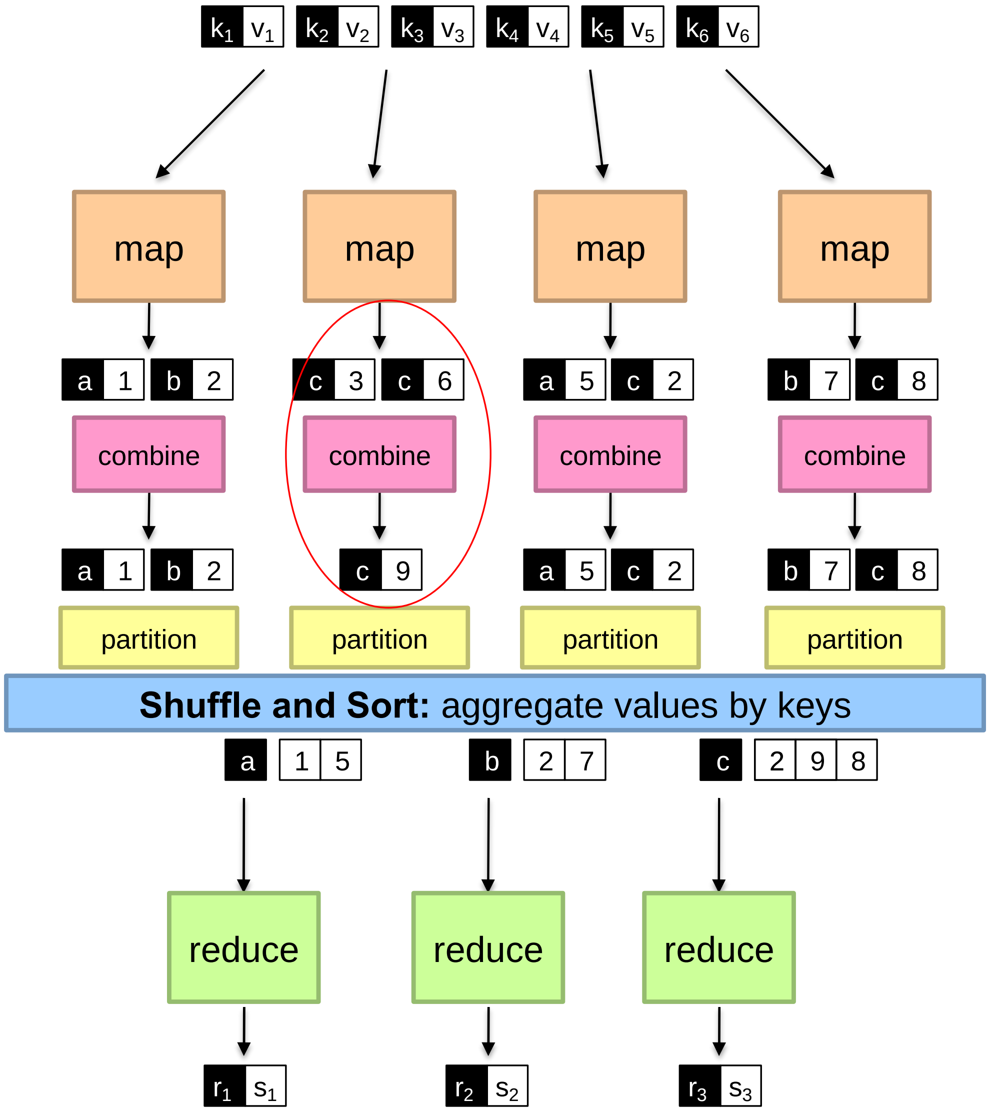
Tutorial 3: MapReduce

Li Yuan

li.yuan@u.nus.edu

Review


k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Question 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** $(p, [i_1, i_2, \ldots, i_k])\ is\ (p, i_1 + i_2 + \cdots + i_k)$.

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

Solution 1

- *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)

Question 2

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

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

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

Solution 2

Section 2.3.1: Matrix-Vector Multiplication by MapReduce

- Suppose we have an $n \times n$ matrix M, whose element in row i and column j will be denoted m_{ij}. Suppose we also have a vector v of length n, whose jth element is v_j. Then the matrix-vector product is the vector x of length n, whose ith element x_i is given by $\boldsymbol{x_i = \sum_{j=1}^{n} m_{ij} \cdot v_j}$.

- The Map Function: for each matrix element m_{ij}, produce the key-value pair (i,$m_{ij} \cdot v_j$)

- The Reduce Function: sum all the values associated with a given key i, and the result is a pair (i,x_i)

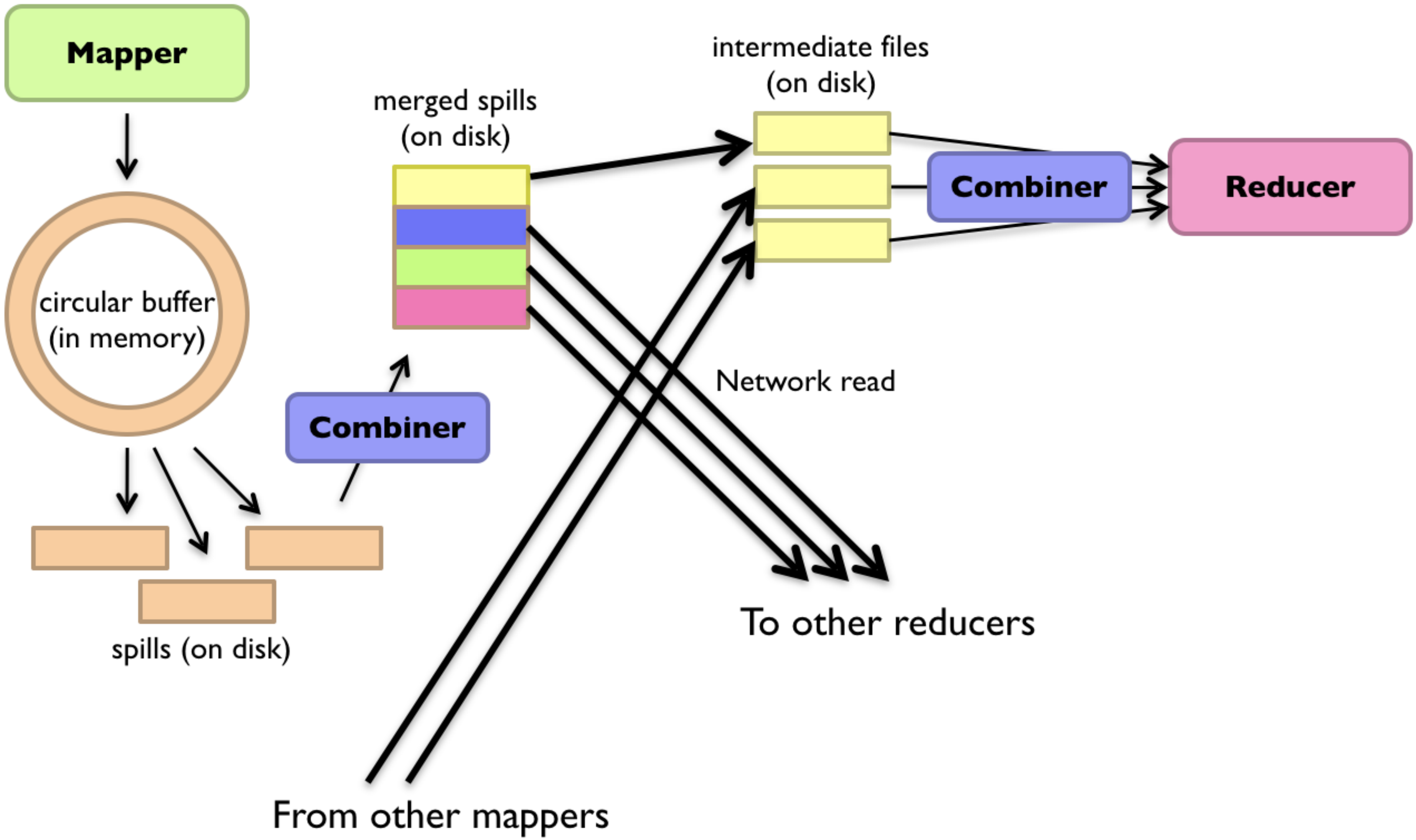
Solution 2

- Thus, in row-major order, the sixteen key-value pairs produced are:

$$\begin{bmatrix} (1,1) & (1,4) & (1,9) & (1,16) \\ (2,5) & (2,12) & (2,21) & (2,32) \\ (3,9) & (3,20) & (3,33) & (3,48) \\ (4,13) & (4,28) & (4,45) & (4,64) \end{bmatrix}$$

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

Shuffle and Sort



Question 3

- **Consider a simple example:**
  - We have a large dataset where input keys are strings and input values are integers.
  - We wish to compute the mean 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 program Tommy has implemented the problem on MapReduce. He has written a few versions with the pseudo code shown in Figures 1—4.

Question 3a

```
class MAPPER
    method MAP(string t, integer r)
        EMIT(string t, integer r)
```

```
class REDUCER
    method REDUCE(string t, integers [r1, r2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r1, r2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        r_avg ← sum/cnt
        EMIT(string t, integer r_avg)
```

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?

Solution 3a

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

Question 3b

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

class Combiner
    method Combine(string t, integers [r1, r2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r1, r2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        Emit(string t, pair (sum, cnt))                    ▷ Separate sum and count

class Reducer
    method Reduce(string t, pairs [(s1, c1), (s2, c2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
```

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?

Solution 3b

- **Combiners are optimizations that cannot change the correctness of the algorithm.**
- **Combiner must have the same input and output key-value type**
- **If Combiner removed, 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.**

Question 3c

```
class Mapper
    method Map(string t, integer r)
        Emit(string t, pair (r, 1))

class Combiner
    method Combine(string t, pairs [(s1, c1), (s2, c2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        Emit(string t, pair (sum, cnt))

class Reducer
    method Reduce(string t, pairs [(s1, c1), (s2, c2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        r_avg ← sum/cnt
        Emit(string t, pair (r_avg, cnt))
```

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).

Solution 3c

- 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).

Question 3d

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

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?

Solution 3d

- 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.

Summary

- **Mapper and Reducer is the key operation of divide and conquer.**
- **Combiner is an optimization step to reduce the amount of data transmission.**
- **Leverage the design of MapReduce program (How to partition the data).**

Acknowledgement

Thanks to Li Qinbin for making these slides.

liqinbin@u.nus.edu