Combiners in MapReduce

MapReduce Job Components

- Input path (set of directories and files)
- Output path (output directory)
- map() function
- reduce() function
- combine() [OPTIONAL]

MapReduce Job Components

- map() function
  - accept a (key, value)
  - convert (key, value) to a set of (key2, value2) pairs
- reduce() function
  - accept (key2, [V_1, V_2, ..., V_n])
  - convert (key2, [V_1, V_2, ..., V_n])
    to a set of (key3, value3) pairs

Typical MapReduce Job

A typical MapReduce job has two functions:

- **`map()`**
- **`reduce()`**

But you can further optimize the output of mappers by adding a combiner function (works very similar to the **`reduce()`** function):

- **`combine()`**

What is a Combiner?

- **`combine() [OPTIONAL]`**
- Combiner is also known as “Mini-Reducer” that summarizes the mappers output with the same key before passing to the Reducer.
- The primary job of Combiner is to process the output data from the mappers, before passing it to reducer.
- The combine() function runs after the mapper and before the reducer

Informal Example

- **Mappers output:**

```
Partition-1: (K, v1), (K, v2), (K, v3)
Partition-2: (K, t1), (K, t2), (K, t3), (K, t4)
```

Rather than sending (K, [v1, v2, v3, t1, t2, t3, t4]) to a reduce() function, we can send (K, [V, T])

Where:

```
V = combine([v1, v2, v3])
T = combine([t1, t2, t3, t4])
```

Informal Example

- Mappers output:

Partition-1: (K, v1), (K, v2), (K, v3)

Partition-2: (K, t1), (K, t2), (K, t3), (K, t4)

Rather than sending (K, [v1, v2, v3, t1, t2, t3, t4]) to a reduce() function,
we can send (K, [V, T])

where

V = combine([v1, v2, v3])

T = combine([t1, t2, t3, t4])

Note that you have to guarantee 4 properties:

1. Type(V) = Type(v1) = Type(v2) = Type(v3)
2. Type(T) = Type(t1) = Type(t2) = Type(t3) = Type(t4)
3. combine() MUST be a **commutative** function.
4. combine() MUST be an **associative** function.

Combiner Example: find sum of values per key

- **Mappers output:**

Partition-1: (K, 2), (K, 3), (K, 4)

Partition-2: (K, 5), (K, 6), (K, 7), (K, 8)

Rather than sending (K, [2, 3, 4, 5, 6, 7, 8]) to a reduce() function, we can send (K, [9, 26]), where

```
 9 = combine([2, 3, 4])
26 = combine([5, 6, 7, 8])
```

Note the the addition (+) is a commutative and an associative function.

What about Combiners?

Combiner is also known as "**Mini-Reducer**" that summarizes the mapper output record with the **same Key** before passing to the Reducer.

`Mappers → Combiners → Reducers`

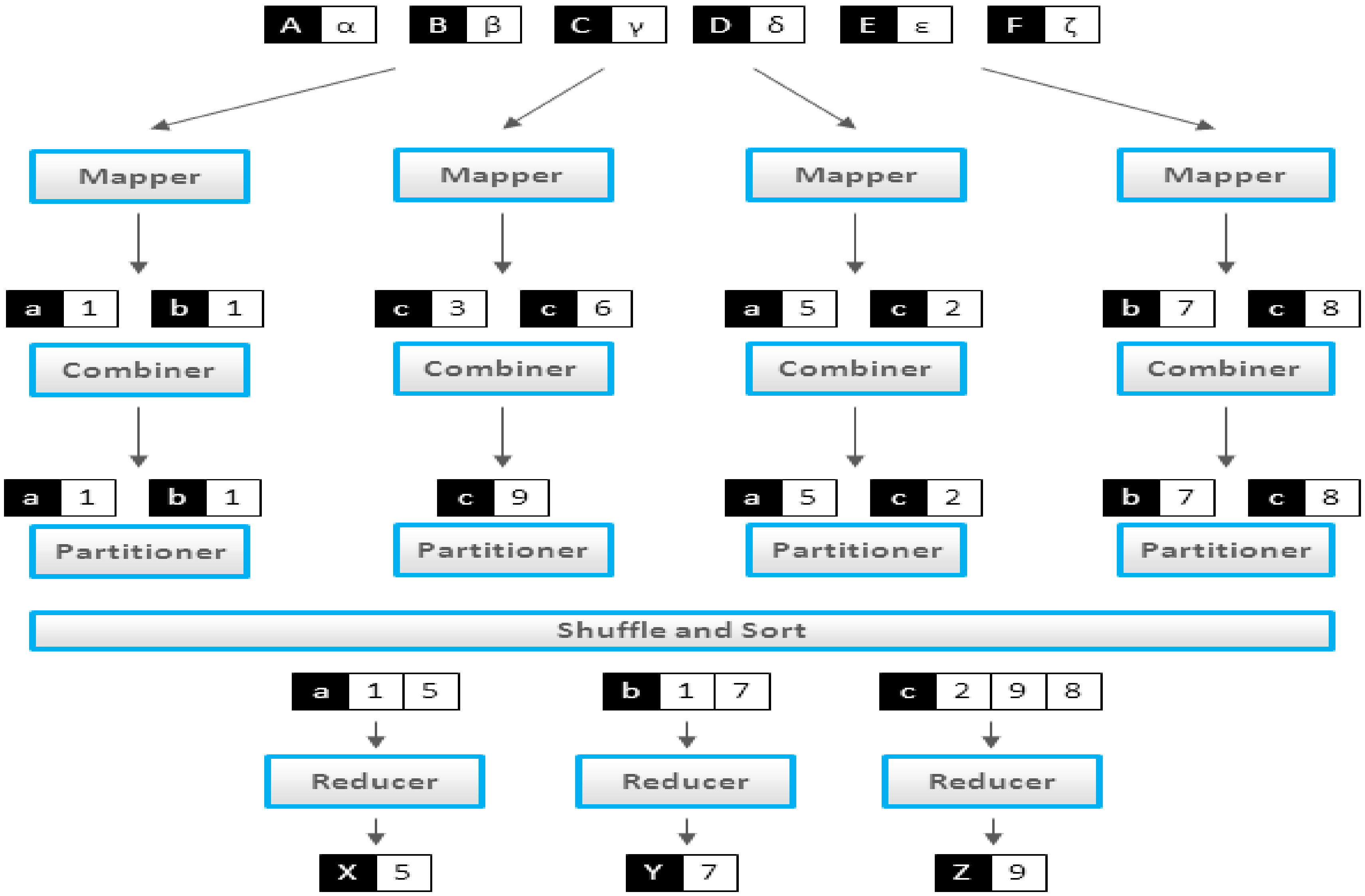
Combiner Example

In the following figure (next slide), for the 2nd partition, mappers have created:

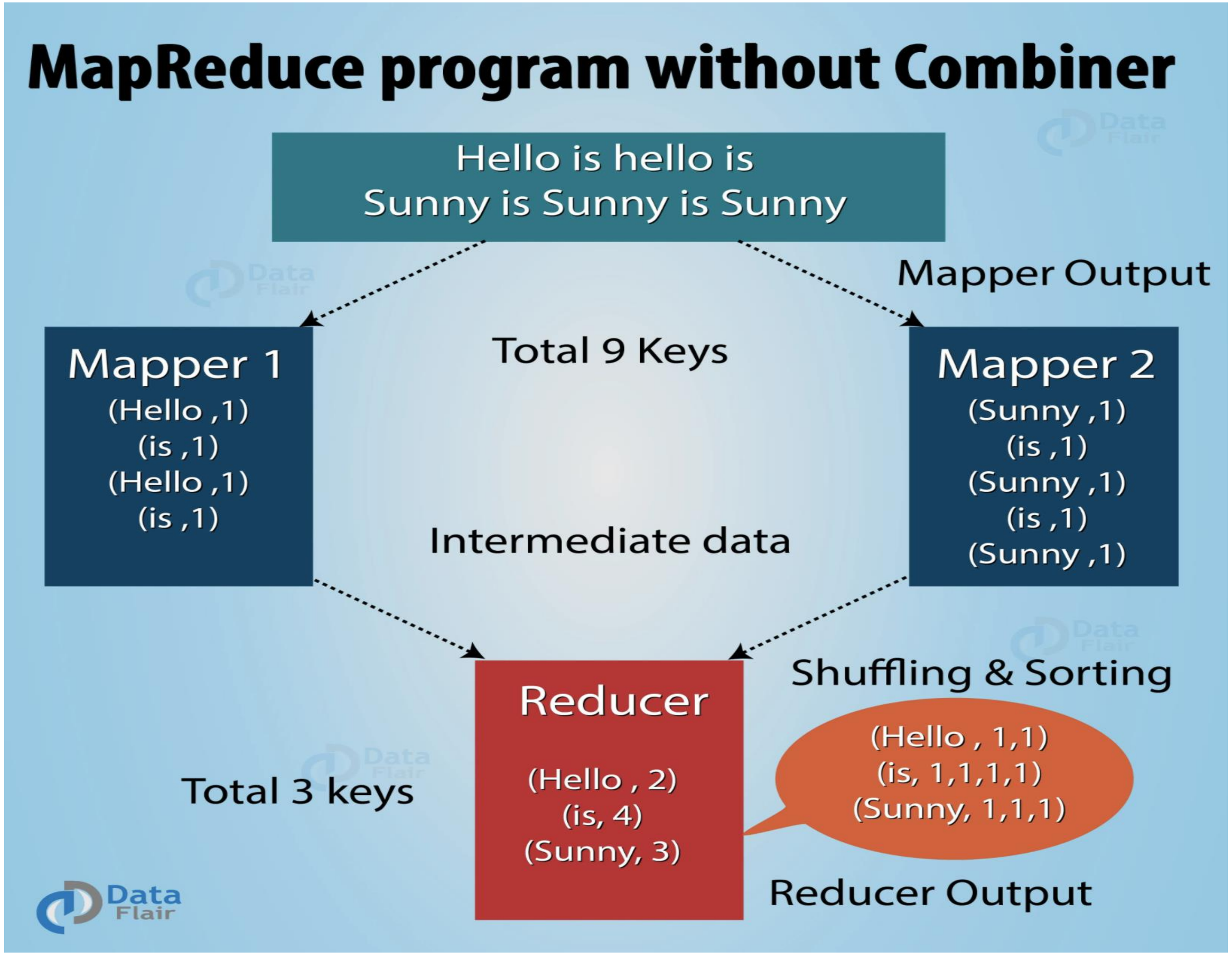
`(c, 3)`
`(c, 6)`

The combiner function combines these two (with the SAME key as "c") into **`(c, 9)`**, `where` **`9 = 3 + 6`**

What about Combiners?



MapReduce without Combiners



Combiner Example

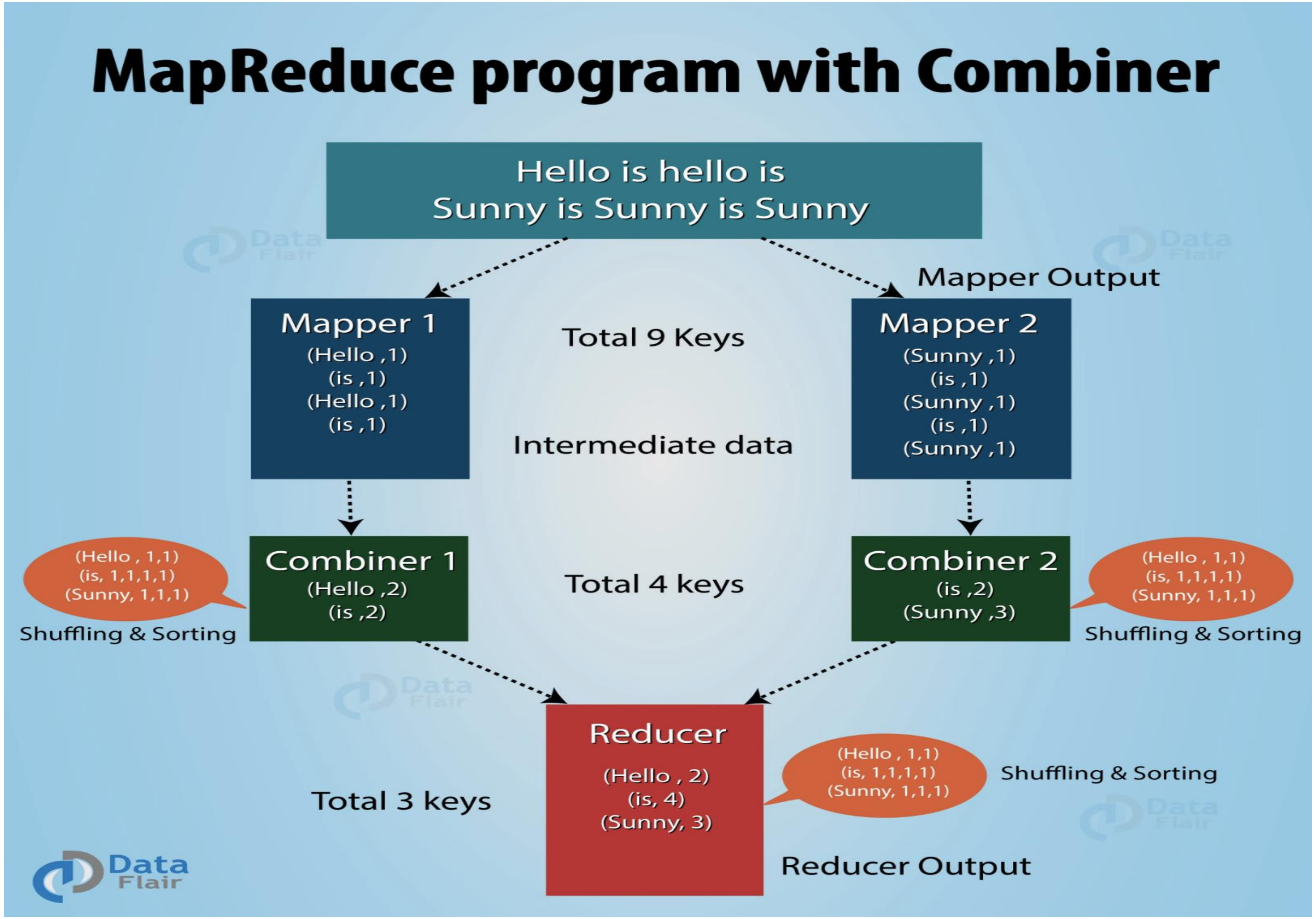
In the following figure (next slide),

Mapper 1	Mapper 2
`(Hello, 1)`	`(Sunny, 1)`
`(Hello, 1)`	`(Sunny, 1)`
	`(Sunny, 1)`

The combiner function combines these two into

`(Hello, 2)` `(Sunny, 3)`

MapReduce with Combiners



EXAMPLE-1: Find Sum of Values per Key

Solution-1: Without Combiners

- map()
- reduce()

Solution-2: With Combiners

- map()
- combine()
- reduce()

EXAMPLE-1: Find Sum of Values (ratings) per Key (movie_id)

Solution-1: Without Combiners

```
# K: record number: ignored
# V: record as “<movie_id><,><rating>”
map(K, V) {
  tokens = V.split(“,”)
  movie_id = tokens[0]
  rating = int(tokens[1])
  emit (movie_id, rating)
}
```

EXAMPLE-1: Find Sum of Values (ratings) per Key (movie_id)

Solution-1: Without Combiners

Sort & Shuffle phase will produce:

```
(movie_id_1, [2, 4, 5, 1, 1, 3])
(movie_id_2, [1, 1, 3, 5])
(movie_id_3, [1, 1, 1, 1, 2, 2])
…
```

EXAMPLE-1: Find Sum of Values (ratings) per Key (movie_id)

Solution-1: Without Combiners

```
# K: a unique movie_id
# V: [v_1, v_2, …, v_n] (all ratings for K)
# V denotes all ratings for K (a unique
movie_id)
reduce(K, V) {
  sum_of_ratings = sum(V)
  emit (K, sum_of_ratings)
}
```

EXAMPLE-1: Find Sum of Values per Key

Solution-2: With Combiners

- map()
- combine()
- reduce()

EXAMPLE-1: Find Sum of Values (ratings) per Key (movie_id)

Solution-2: With Combiners

```
# K: record number: ignored
# V: record as “<movie_id><,><rating>”
map(K, V) {
  tokens = V.split(“,”)
  movie_id = tokens[0]
  rating = int(tokens[1])
  emit (movie_id, rating)
}
```

EXAMPLE-1: Find Sum of Values (ratings) per Key (movie_id)

Solution-2: With Combiners

combine() function combines output of mappers per worker node for the same key:

```
# K: a unique movie_id
# V: [v_1, v_2, …, v_n]
# V denotes all ratings for K
combine(K, V) {
  sum_of_ratings = sum(V)
  emit (K, sum_of_ratings)
}
```

EXAMPLE-1: Find Sum of Values (ratings) per Key (movie_id)

Solution-2: With Combiners

```
reduce(): reducer function

# K: a unique movie_id
# V: [v_1, v_2, …, v_n]
# V denotes all ratings for K
reduce(K, V) {
  sum_of_ratings = sum(V)
  emit (K, sum_of_ratings)
}
```

How do we write Combiners? For Averages?

We need to write 3 functions:

- map()
- combine()
- reduce()

- BUT Note that
  - “average of an average is not an average”
- What does this mean?

Average of an Average is not an Average

- Let say we have 2 partitions
- Partition-1: (K, 6), (K, 7)
  - Average of Partition-1: (6+7)/2 = 6.5
- Partition-2: (K, 8)
  - Average of Partition-2: (8)/1 = 8.0
- Average of Partition-1 and Partition-2:
  - (6.5 + 8.0)/2 = 7.25 => **NOT CORRECT**

Average(6, 7, 8) = (6+7+8)/3 = 21/3 = 7.0

Hmmmmmm? How to solve this?

Make Average of an Average as an Average By Changing Output of Mappers

- Let say we have 2 partitions
- Partition-1: (K, 6), (K, 7)
- Partition-2: (K, 8)
- (K, V) → (K, (V, 1))

- Change map() to create **(K, (sum, count))**
- (K, 6) --> (K, (6, 1))
- (K, 7) --> (K, (7, 1))
- (K, 8) --> (K, (8, 1))

Make Average of an Average as an Average

- Let say we have 2 partitions:
- Partition-1: (K, (6, 1)), (K, (7, 1))
- Partition-2: (K, (8, 1))

- Average(Partition-1): (K, (6+7, 1+1)) = (K, (13, 2))
- Average(Partition-2): (K, (8, 1))

- **Average**(Partition-1, Partition-2) **=**

(K, (13+8, 2+1)) =
(K, (21, 3)) =
(K, (sum, count))
=> (K, 21/3) = (K, 7)

Sample output of Mappers

- Let say we have 2 partitions:
  - Partition-1: (K, (6, 1)), (K, (7, 1))
  - Partition-2: (K, (8, 1))

- combine(Partition-1): (K, (6+7, 1+1)) = (K, (13, 2))
- combine(Partition-2): (K, (8, 1))

Combine must be Associative & Commutative

Commutative:

(a + b) = (b + a)

```
F(a, b) = F(b, a)
(sum1, count1) + (sum2, count2) = (sum2, count2)  + (sum1, count1)
(sum1+sum2, count1+count2) = (sum2+sum1, count2+count1)
                           = (sum1+sum2, count1+count2)
```

Associative:

(a + (b + c)) = ((a + b) + c)

```
F(a, F(b, c)) = F(F(a, b), c)
(sum1, count1) + ((sum2, count2) + (sum3, count3)) =
((sum1, count1) + (sum2, count2)) + (sum3, count3)
```

What to consider for combiners & reducers

- Make their functions to be associative and commutative:
- Let + be a binary function

- Commutative Laws

$$a + b = b + a$$

- Associative Laws

$$(a + b) + c = a + (b + c)$$

Commutative Example

- Addition is commutative
  - 2 + 3 = 3 + 2 = 5
  - 100 + 200 = 200 + 100 = 300

- Multiplication is commutative
  - 2 * 5 = 5 * 2 = 10
  - 20 * 30 = 30 *20

Subtraction and Division is NOT Commutative

<u>Subtraction</u>

- F (a, b) != F(b, a)
- 5 - 3 = 2
- 3 - 5 = -2
- 2 NOT EQUAL to -2

<u>Division</u>

- 10 / 2 = 5
- 2 / 10 = 0.2
- 5 NOT EQUAL to 0.2

Average function is not Associative

- Avg (1, 2, 3) = 2.0
- Avg(1, Avg(2, 3)) = Avg(1, 2.5) = 1.75
- 2.0 NOT EQUAL to 1.75

References

1. Monoidify! Monoids as a Design Principle for Efficient MapReduce Algorithms by Jimmy Lin

2. Data Algorithms (book) by Mahmoud Parsian

3. Data Algorithms with Spark (book) by Mahmoud Parsian