Map/ Reduce

Nora Steil

History of MapReduce

- Jeffrey Dean and Sanjay Ghemawat 2004
- Used by Google until 2014
- Java-based distributed execution framework
  - Map
  - Combine & Partition
  - Reduce
- Processes and generates large data sets

Important Notes

- Assume that we are using a Distributed File System
- No data movement
- Key-Value structure
- Handling machine failures
- Map and Reduce are idempotent

Step 1: Map

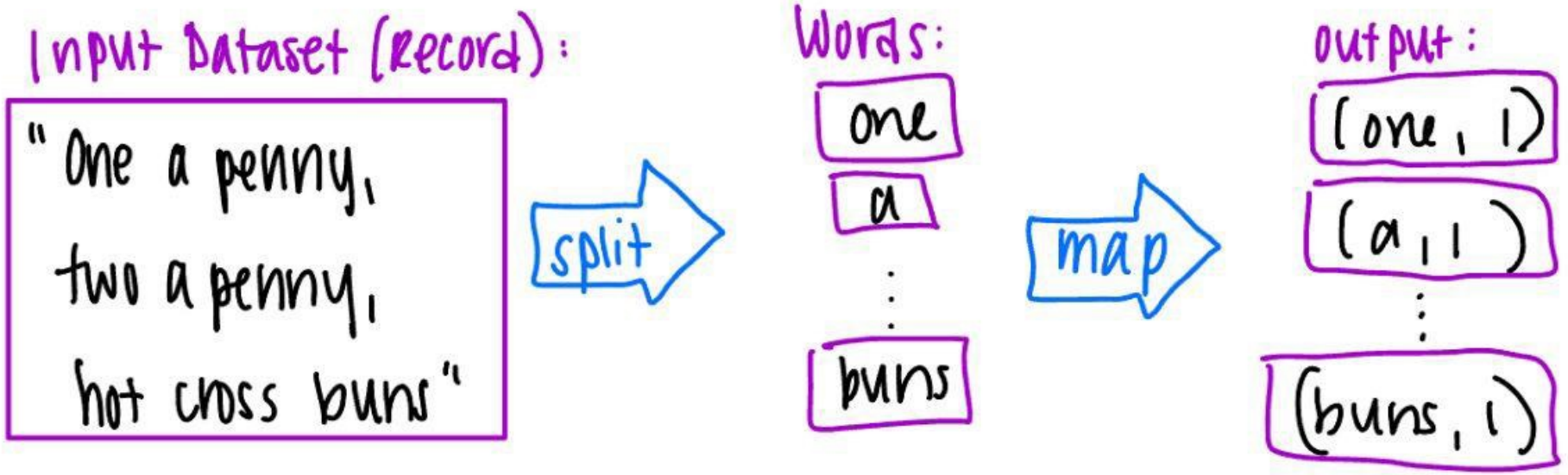
- Splits input “records” into (key, value) pairs
- Outputs a set of (key, value) pairs
- Groups values associated with the same key
    - Values passed to Reduce
- Input: (K, V)
- Output: (K, V)

Word Count Example - Map Step

“One a penny, two a penny, hot cross buns”

```
map(String record) {
        For each word in record
                emit(word, 1).
}
```

(“one”, 1), (“a”, 1), (“penny”, 1), (“two”, 1), (“a”, 1), (“penny”, 1), (“hot”, 1), (“cross”, 1), (“buns”, 1)


Input Dataset (record):
"One a penny,
two a penny,
hot cross buns"
split
Words:
one
a
buns
map
output:
(one, 1)
(a, 1)
(buns, 1)

Step 2: Combine and Partition

- Combine often performed as a part of the reduce step
  - Can make reduce easier
  - Puts all data for one key in the same node
- Partition is not optional
  - Determines how to present data to reducer
  - Assigns data to a reducer
- Exchanges data between machines
- Input: (K, V) from map
- Output: (K, list(V))

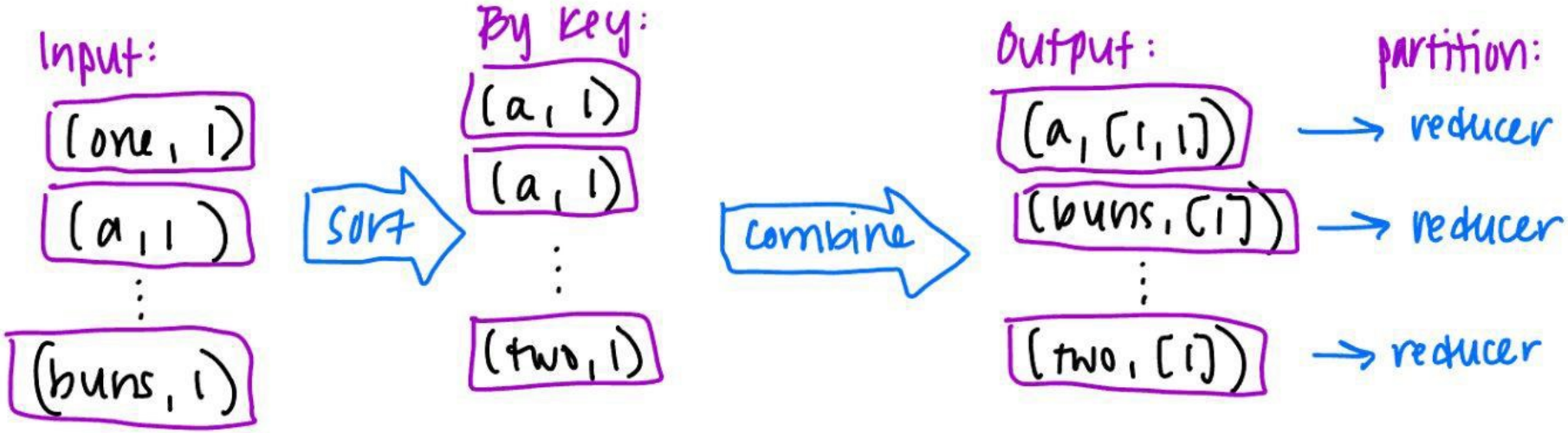
Word Count Example - Combine & Partition

(K, V) input from Map:

(“one”, 1), (“a”, 1), (“penny”, 1), (“two”, 1), (“a”, 1), (“penny”, 1), (“hot”, 1), (“cross”, 1), (“buns”, 1)

(K, list(V)) output to send to Reduce:

(“a”, [1,1]), (“buns”, [1]), (“cross”, [1]), (“hot”, [1]), (“one”, [1]), (“penny”, [1,1]), (“two”, [1])


Input:
(one, 1)
(a, 1)
(buns, 1)
sort
By key:
(a, 1)
(a, 1)
(two, 1)
combine
Output:
(a, [1, 1])
(buns, [1])
(two, [1])
partition:
reducer
reducer
reducer

Step 3: Reduce

- Adds up values from Combine & Partition
- Input: (K, list(V))
- Output: (K, V)
- Reduce output (K, V) is different from Map input (K, V)

```
reduce(String key, List value_list) {
        String word = key;
        int count = 0;
        For each value in value_list
                count = count + value
        output(word, count)
}
```

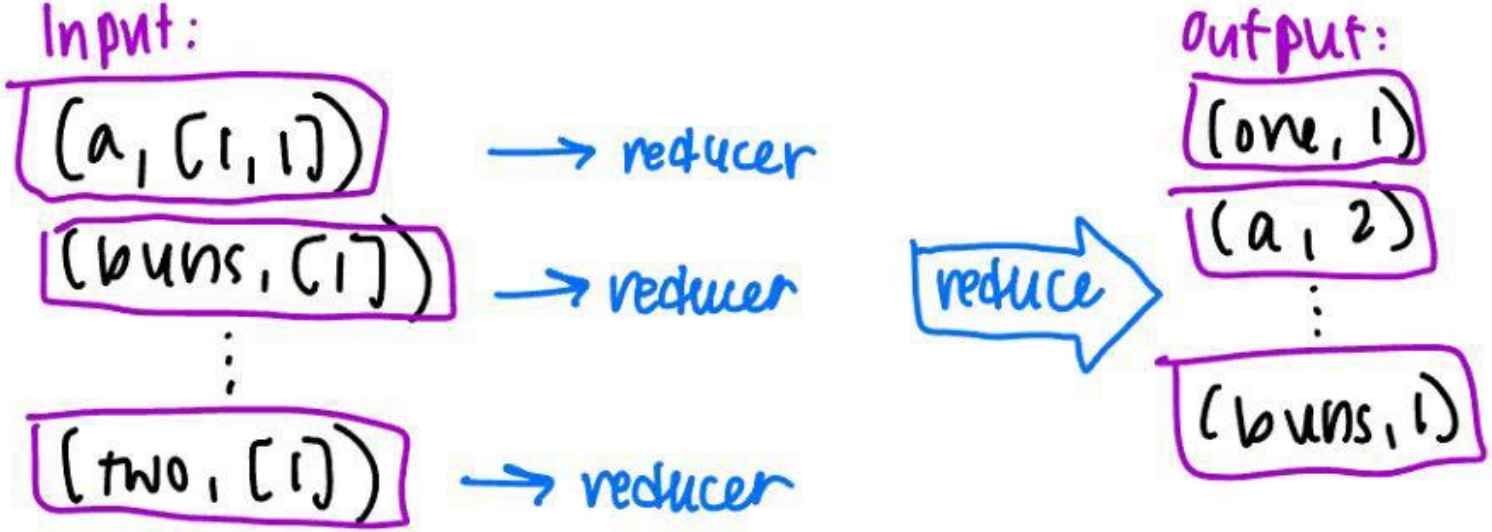
Word Count Example - Reduce

(K, list(V)) input from Combine & Partition:

(“a”, [1,1]), (“buns”, [1]), (“cross”, [1]), (“hot”, [1]), (“one”, [1]), (“penny”, [1,1]), (“two”, [1])

(K, V) output:

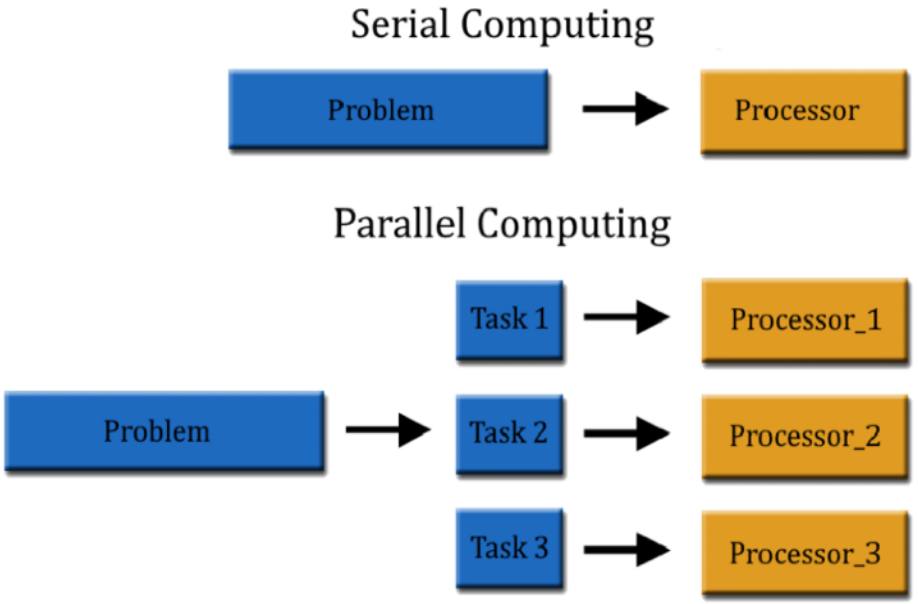
(“one”, 1), (“a”, 2), (“penny”, 2), (“two”, 1), (“hot”, 1), (“cross”, 1), (“buns”, 1)


Input:
(a, [1, 1])
(buns, [1])
(two, [1])
reducer
reducer
reducer
reduce
Output:
(one, 1)
(a, 2)
(buns, 1)

Phase	**Input**	**Output**
Map	(K, V)	(K, V)
Combine & Partition	(K, V)	(K, list(V))
Reduce	(K, list(V))	(K, V)

Parallel Processing

- Data broken into pieces
  - Each piece processed simultaneously
- Faster than serial processing
- Map and Reduce both run in parallel
- Create copies to prevent data loss



Hadoop

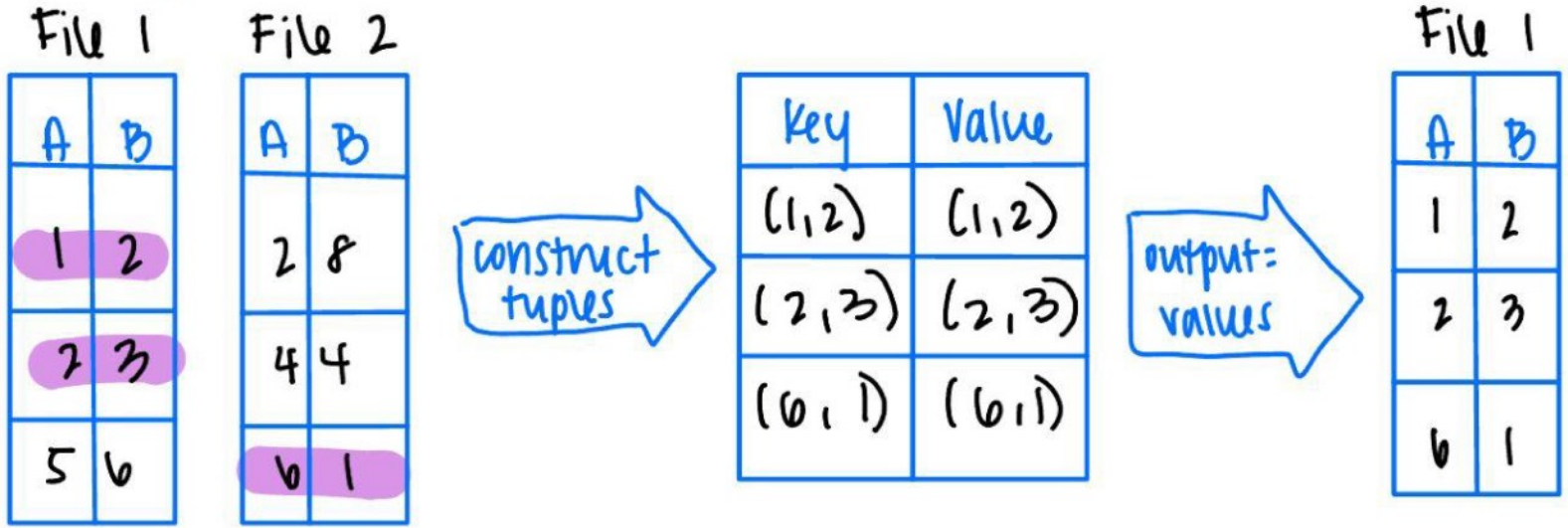
- Library implementation of MapReduce
  - Java
- Executed on server where data resides
- Can take many formats of input files
  - Automatically breaks up some files
- Allows storing (key, value) pairs in MongoDB
- Map and Reduce functions created for programmer
  - Mapper and Reducer
- Combine() optional

Hadoop Input and Output

- Mapper and Reducer take 4 type arguments
  - Specified by programmer
  - reduce() input key same as map() output
  - reduce() input value iterable

SQL and MapReduce: Select

- One map() function
- Each row is a record
  - Outputs key value pair if row meets condition
- Reduce is not used


selecting B<=3
File 1
A
B
1
2
2
3
5
6
File 2
A
B
2
8
4
4
6
1
construct tuples
Key
Value
(1,2)
(1,2)
(2,3)
(2,3)
(6,1)
(6,1)
output: values
File 1
A
B
1
2
2
3
6
1

SQL and MapReduce: Group By & Aggregation

- Map implicitly groups keys
  - Key is attributes being grouped
  - Value is aggregated values
- Reduce implicitly aggregates
  - Apply aggregation operation on values

Group by A, B
Sum

Map Worker 1

File 1

A	B	C	D
1	2	3	1
2	2	3	2
1	2	1	3

File 2

A	B	C	D
4	2	1	3
6	8	4	4
3	2	2	4

Map Worker 2

File 1

A	B	C	D
1	2	5	2
2	3	2	4
1	3	1	3

File 2

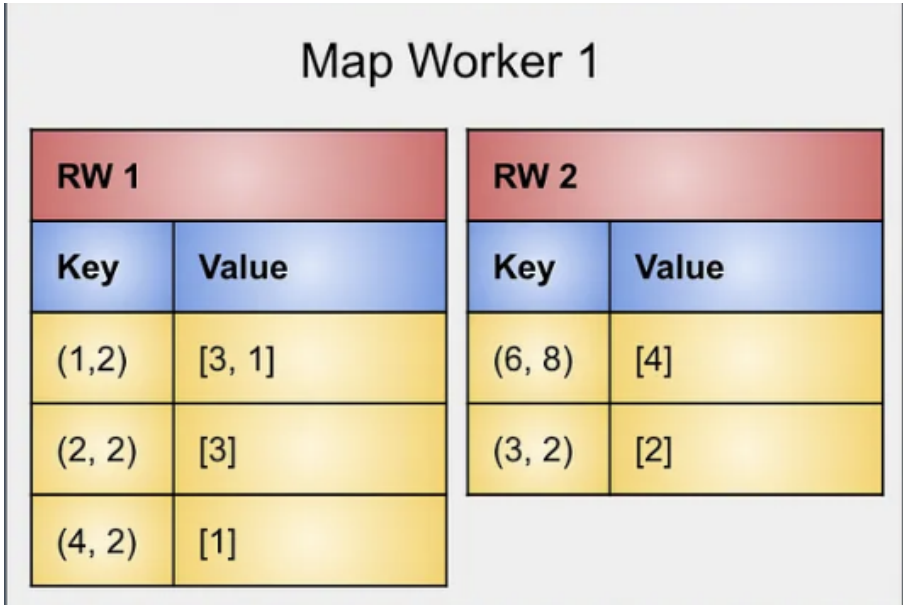
A	B	C	D
3	2	1	3
2	3	9	2
3	4	2	1

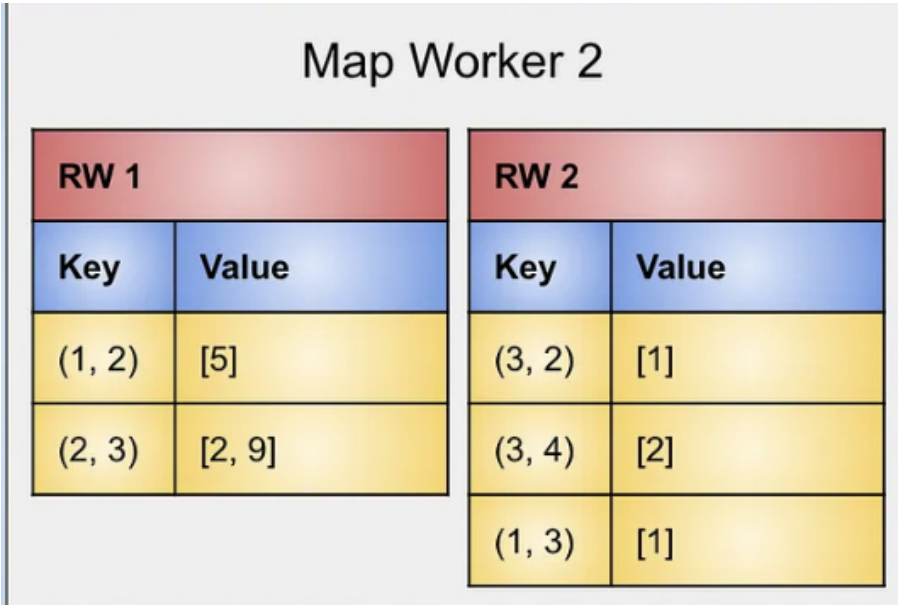
Map Worker 1

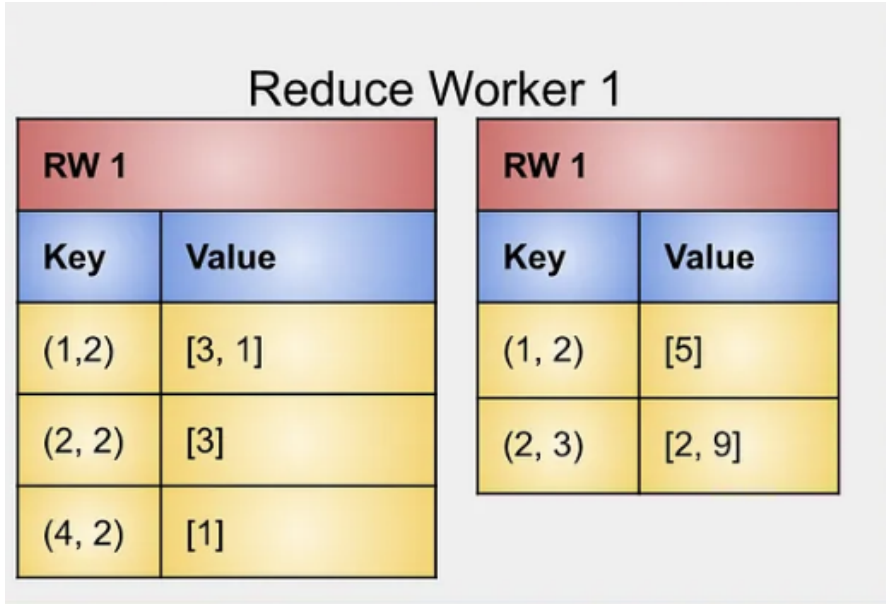
Key	Value
(1, 2)	[3, 1]
(2, 2)	[3]
(4, 2)	[1]
(6, 8)	[4]
(3, 2)	[2]

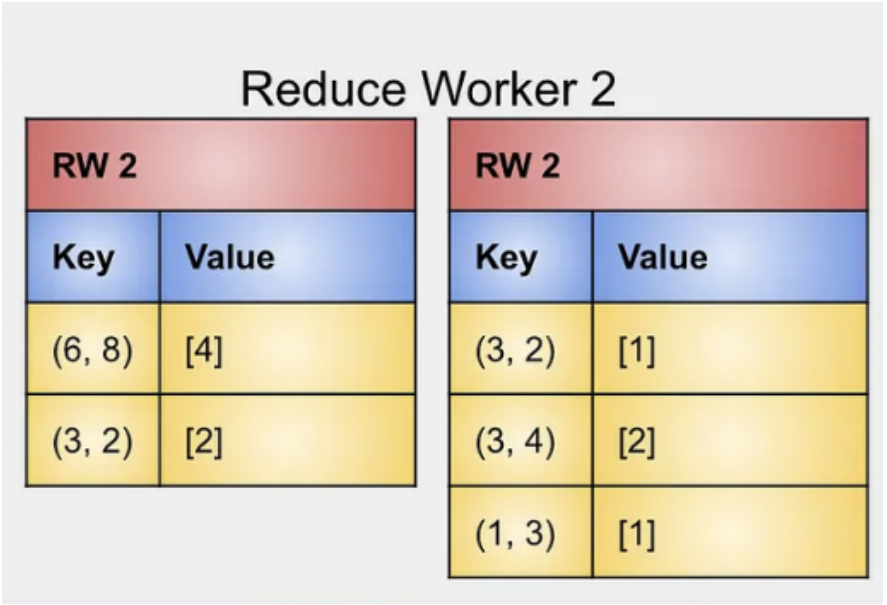
Map Worker 2

Key	Value
(1, 2)	[5]
(2, 3)	[2, 9]
(1, 3)	[1]
(3, 2)	[1]
(3, 4)	[2]


Map Worker 1
RW 1
Key
Value
(1,2)
[3, 1]
(2, 2)
[3]
(4, 2)
[1]
RW 2
Key
Value
(6, 8)
[4]
(3, 2)
[2]


Map Worker 2
RW 1
Key
Value
(1, 2)
[5]
(2, 3)
[2, 9]
RW 2
Key
Value
(3, 2)
[1]
(3, 4)
[2]
(1, 3)
[1]


Reduce Worker 1
RW 1
Key
Value
(1,2)
[3, 1]
(2, 2)
[3]
(4, 2)
[1]
RW 1
Key
Value
(1, 2)
[5]
(2, 3)
[2, 9]


Reduce Worker 2
RW 2
Key
Value
(6, 8)
[4]
(3, 2)
[2]
RW 2
Key
Value
(3, 2)
[1]
(3, 4)
[2]
(1, 3)
[1]

Reduce Worker 1

Key	Value
(1,2)	[3, 1, 5]
(2, 2)	[3]
(4, 2)	[1]
(2, 3)	[(2, 9)]

Reduce Worker 2

Key	Value
(6, 8)	[4]
(3, 2)	[1, 2]
(3, 4)	[2]
(1, 3)	[1]

Reduce Worker 1

A	B	Sum
1	2	9
2	2	3
4	2	1
2	3	11

Reduce Worker 2

A	B	Sum
6	8	4
3	2	3
3	4	2
1	3	1

SQL and MapReduce Summary

SQL Clause	SELECT	GROUP BY
map()	Yes	Yes
reduce()	Only with GROUP BY, SUM(), or COUNT()	Yes

Sources

Silberschatz 10.3

https://www.tutorialscampus.com/map-reduce/algorithm.htm

https://www.todaysoftmag.com/article/1358/hadoop-mapreduce-deep-diving-and-tuning

https://informationit27.medium.com/hadoop-mapreduce-partition-in-hadoop-administration-fdf265cd1eaa

https://www.youtube.com/watch?v=cHGaQz0E7AU

https://medium.com/swlh/relational-operations-using-mapreduce-f49e8bd14e31