ETH zürich

Week 7 - MapReduce

Aljaž Medič

Plan for today

1. Exercise Sheet Review & Questions
2. Quiz
3. MapReduce overview & examples
4. Old Exam Questions

Exercise Sheet Review 6 - Task 2.1

Document 1 — Doc 1 - 4

```
<happiness xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="Schema.xsd"/>
```

Document 2 — Doc 2 - 1, 2

```
<happiness xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="Schema.xsd">
    <health/>
    <friends/>
    <family/>
</happiness>
```

Document 3 — Doc 3 - 3

```
<happiness xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="Schema.xsd">
    3.141562
</happiness>
```

Document 4 — Doc 4 - 1, 2, 5

```
<happiness xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="Schema.xsd">
    <health value="100"/>
    <friends/>
    <family/>
</happiness>
```

Document 5 — Doc 5 - 2

```
<happiness xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="Schema.xsd">
    <health/>
    <friends/>
    <family/>
    But perhaps everybody defines it differently...
</happiness>
```

5

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="happiness">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="health">
                    <xs:complexType>
                        <xs:attribute name="value" type="xs:integer" use="required"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="friends"/>
                <xs:element name="family"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="happiness" type="xs:decimal"/>
</xs:schema>
```

1

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="happiness">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="health"/>
                <xs:element name="friends"/>
                <xs:element name="family"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

4

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="happiness">
        <xs:complexType>
            <xs:sequence/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

2

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="happiness">
        <xs:complexType mixed="true">
            <xs:sequence>
                <xs:element name="health"/>
                <xs:element name="friends"/>
                <xs:element name="family"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

QUIZ

Which of the following components belong to the classic Hadoop MapReduce v1 architecture?

JobTracker; TaskTracker; NameNode; DataNode; YARN ResourceManager

JobTracker, TaskTracker

How are jobs, splits, tasks and slots related?

Each MapReduce job is divided into splits; each split is processed by a task; and each task runs in a slot.

What are the three main phases of a MapReduce job?

Map, Shuffle, Reduce

QUIZ

MapReduce can only process key-value pairs.

True.

One of the optimizations in MapReduce is, that reduce tasks begin emitting final output before all map tasks have completed.

False. (We have to wait for the last map task to be done, and then shuffle, because a map task can influence any of the reduce tasks.)

MapReduce requires that the input data reside in HDFS.

False. (MapReduce can read input from several places e.g., cloud object storage like S3, HDFS, wide-column stores, etc.)

QUIZ

In MapReduce, the intermediate , and output values are collections of key–value pairs, and their key/value types must be identical.

False. (They are all key–values, but the types do not need to match across input, intermediate, and output; it's only common (not required) for intermediate and output types to be the same.)

Map and reduce tasks can both generate any number of key-value pairs (including zero).

True.

A key–value pair cannot be split across two input splits during splitting, but it can be misaligned with the underlying HDFS Blocks.

True.

QUIZ

A combine and map functions are guaranteed to run exactly once each per map task.

False. (Combine function may run zero or multiple times (e.g., spilling, merging), but there is no guarantee it will run at all.)

Combine task runs after Map task and before Reduce task.

False. (There is no combine task! Just combine function.)

During shuffle, pairs with the same key may be sent to different reduce partitions for load balancing.

False. (Partitioning ensures all pairs with the same key go to the same reduce-side partition.)

QUIZ

A map task invokes the map function in parallel over the records of its split.

False. (Calls of the map function within a single task are sequential.)

A reduce slot can execute multiple reduce tasks concurrently.

False. (One reduce slot handles one reduce task at a time, though it can process multiple tasks over time.

If a running map task fails, the entire MapReduce job immediately fails and stops.

False. (MapReduce is fault-tolerant: if a task fails, the JobTracker will reschedule and rerun that task on another node, so the job can still complete.)

QUIZ

The architecture in MapReduce is decentralized.

False. (The architecture is **distributed**, but still **centralized** - it has a master/worker architecture.)

It’s preferable for MapReduce to produce many small files rather than one big one, for practical reasons.

True.

Within a map or reduce phase, parallelism occurs between the slots.

True.

QUIZ

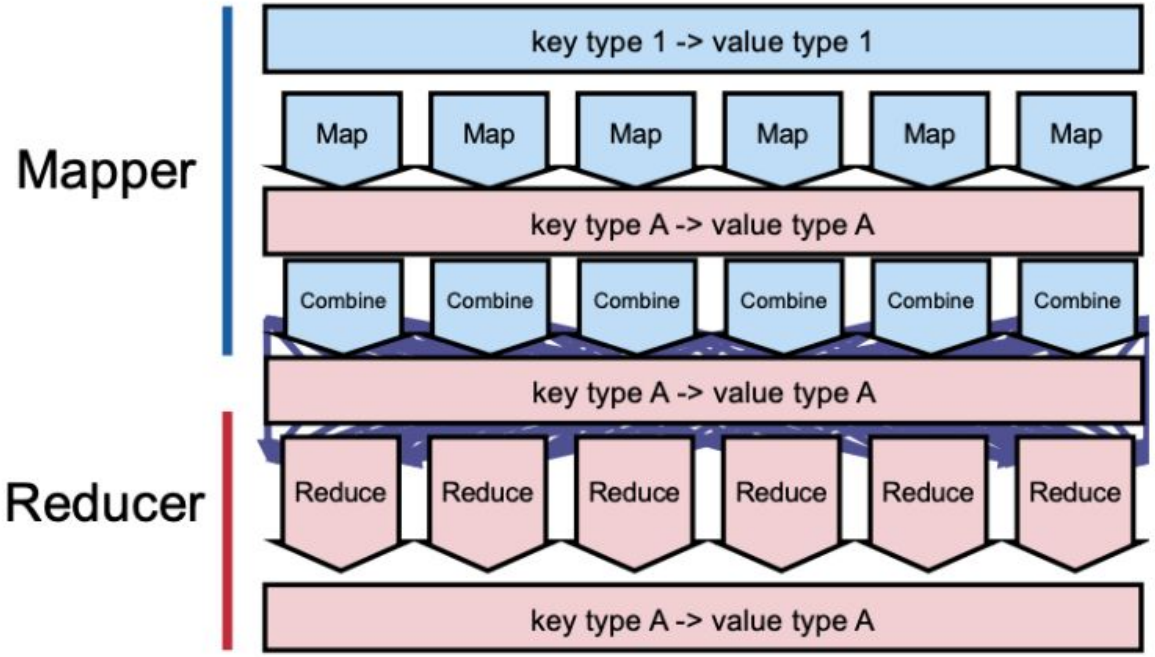
In MapReduce v1, the slot is either declared reduce or map, and cannot change its purpose during the job run, which makes a part of the cluster unused at all stages of MapReduce.

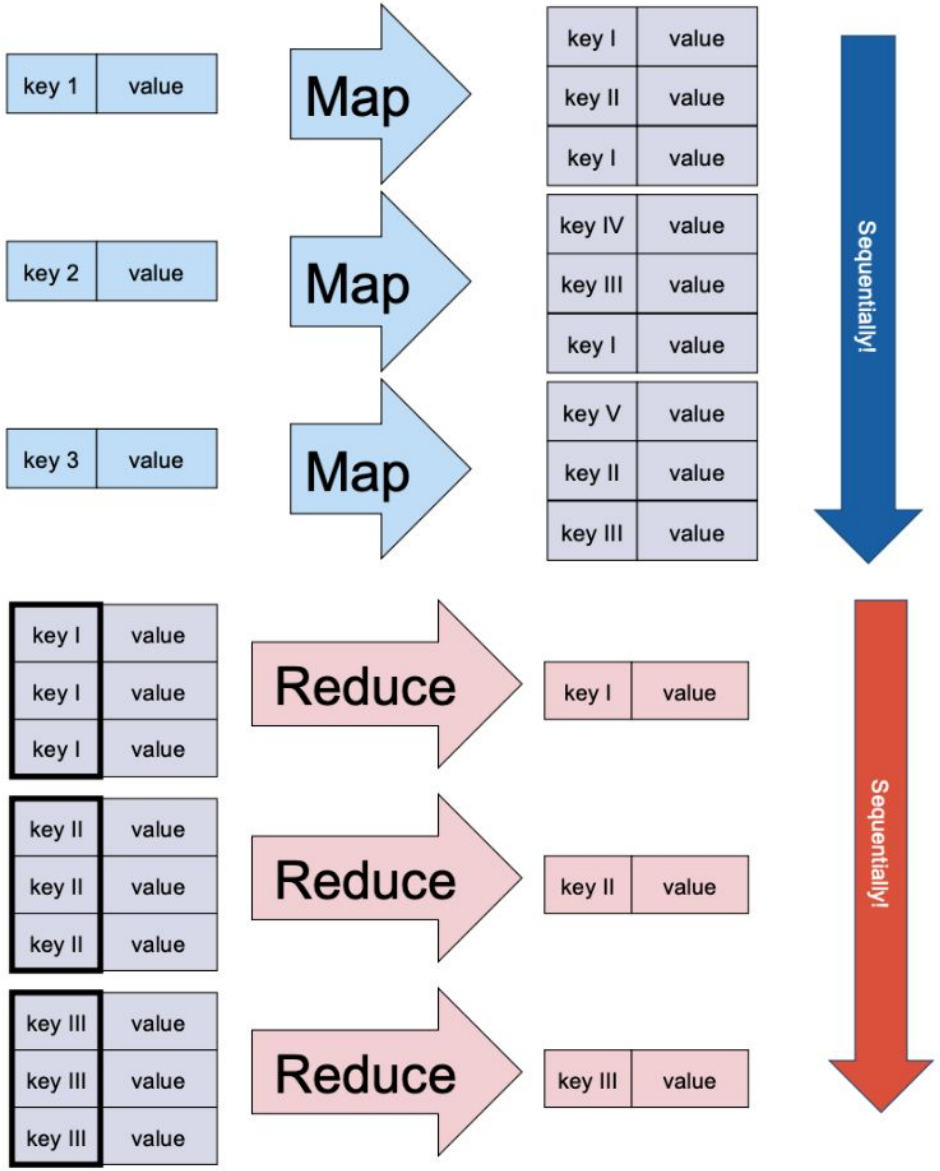
True.

Which properties does a combine function need to have?

It has to be associative and commutative. If this is true for the reduce function, and the key-value types match, we can usually just reuse reduce function.

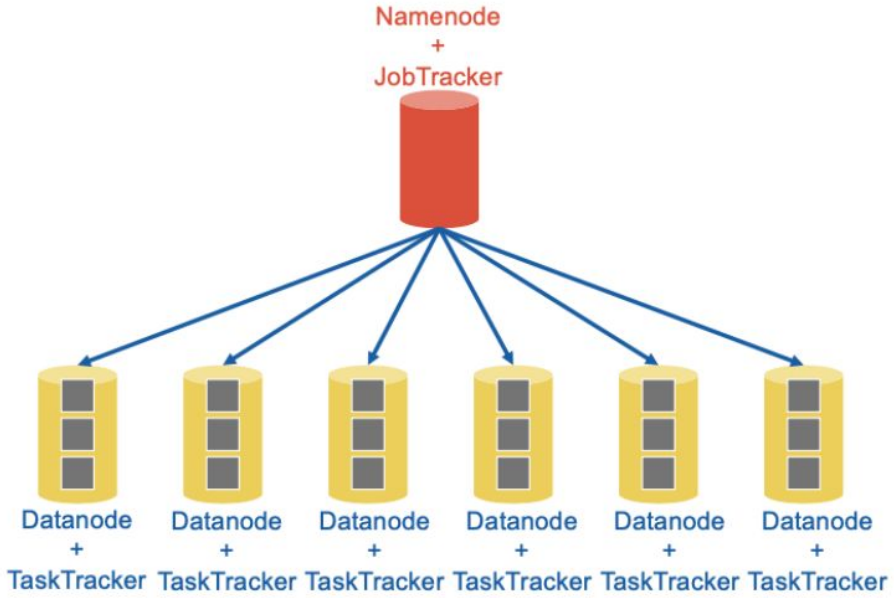
MapReduce

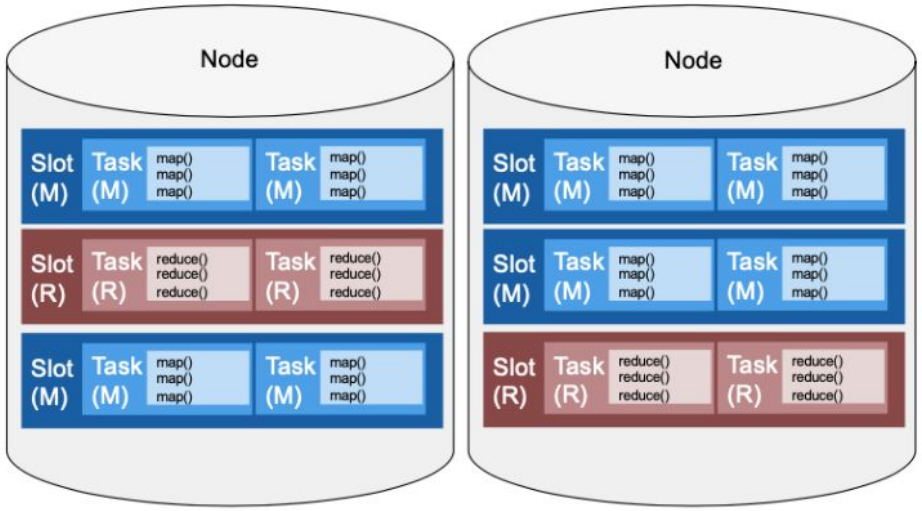

Mapper
Reducer
key type 1 -> value type 1
Map
Map
Map
Map
Map
Map
key type A -> value type A
Combine
Combine
Combine
Combine
Combine
Combine
key type A -> value type A
Reduce
Reduce
Reduce
Reduce
Reduce
Reduce
key type A -> value type A

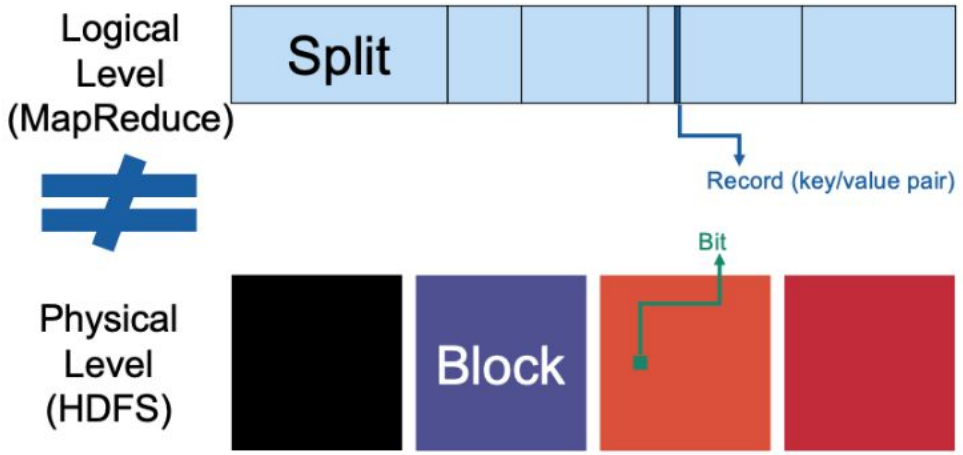

key 1
value
key 2
value
key 3
value
Map
Map
Map
key I
value
key II
value
key I
value
key IV
value
key III
value
key I
value
key V
value
key II
value
key III
value
Sequentially!
key I
value
key I
value
key I
value
key II
value
key II
value
key II
value
key III
value
key III
value
key III
value
Reduce
Reduce
Reduce
key I
value
key II
value
key III
value
Sequentially!

MapReduce - Architecture

- Centralized, distributed architecture.
- We benefit from building on top of HDFS.







Impedance mismatch

Let's Write a Combine function!

```
def map(key, value):
    emit(key, value)

def combine(key, values[]):
    ...

def reduce(key, values[]):
    emit(key, sum(values))
```

```
def combine(key, values[]):
    emit(key, sum(values))
```

Let's Write another Combine function!

```
def map(key, value):
    emit(key, (value, 1))

def combine(key, values[]):
    ...

def reduce(key, values[]):
    sum = 0
    count = 0
    for (val_i, count_i) in values:
        sum += val_i
        count += count_i
    emit(key, sum/count)
```

```
def combine(key, values[]):
    sum = 0
    count = 0
    for (val_i, count_i) in values:
        sum += val_i
        count += count_i
    emit(key, (sum, count))
```

HS23 Q34

```
def reduce(key, values):
    values.sort(reverse=True)
    return values
```

B

```
def reduce(key, values):
    count = 0
    for i in range(1, len(values)):
        if values[i] > values[i - 1]:
            count += 1
    return count
```

D

```
def reduce(key, values):
    return max(values) - min(values)
```

A

```
def reduce(key, values):
    if sum(values) > threshold:
        return key
    return None
```

C

- *(A) Finds the range (difference between max and min) for values of each key.*
- *(B) Groups values by key and sorts them in descending order.*
- *(C) Identifies keys that have values summing to over a specified threshold.*
- *(D) Compares adjacent values for each key and counts the number of increases.*

HS24 Q31

We run a MapReduce job on top of an input dataset:
- whose keys are pairs of date and location, such as ("2025-01-01", "Zurich");
- whose values are non-negative integers, which indicate the number of houses sold on that day and at the given place.

The code is given below:

```
map(key, value):
    if key.date >= "2024-01-01":
        emit(("date", key.date), value)
        emit(("place", key.place), value)

reduce(key, values):
    if key.0 == "date":
        emit(key.1, sum(values))
    else:
        count = 0
        for value in values:
            if value > 0:
                count += 1
        emit(key.1, count)
```

True	False	
	False (task != function call)	Each reduce task will either only process keys that contain a date, or only keys that contain a place.
	False (wrong type)	The reduce function could also be used as a combine function.
True		The code computes for each day, how many houses have been sold since the start of 2024, and for each place, on how many days a house was sold since the start of 2024.
True		If there are many input (day, place) tuples with zero houses sold, the code could be made more efficient by only emitting from the map function when the value is strictly larger than 0.

HS23 Q35

True	False	
○	F	**Scenario**: In a graph represented as a list of edges, a MapReduce job is configured to find the degree of each vertex (i.e., the number of edges connected to each vertex). The map function emits each vertex in an edge with a count of 1. The reduce function sums these counts for each vertex. **Statement**: This MapReduce job requires a secondary sorting mechanism in the reduce phase to correctly calculate the degree of each vertex.
T	○	**Scenario**: Given a large dataset of customer transactions, a MapReduce job is designed to calculate the total expenditure of each customer. The map function emits key-value pairs where the key is the customer ID and the value is the transaction amount. The reduce function sums up all the values for each key. **Statement**: This MapReduce job can benefit from using a combine function to reduce the amount of data transferred across the network.

HS23 Q35

T ○ **Scenario**: A MapReduce job is used for text analysis, where the goal is to count the total number of occurrences of each word in a large collection of documents. The map function emits each word as a key and a count of 1 as the value. The reduce function sums these counts for each word.

Statement: We could adapt the job to compute the number of occurrences of each word per document ("document frequency") by modifying the map function so that it emits composite keys consisting of the document ID and the word.

T ○ **Scenario**: A MapReduce job is implemented to find the minimum and maximum values from a large set of numerical data. The map function processes chunks of data and emits two key-value pairs for each chunk: one for the minimum value and one for the maximum value. The reduce function then finds the global minimum and maximum from these pairs.

Statement: The reduce function in this job is idempotent, meaning that applying it multiple times does not change the outcome beyond the initial application.

ETH zürich

See you next week!

Aljaž Medič
amedic@ethz.ch

Slides

Suggestions