MapReduce Fundamental Concepts

Mata Kuliah Big Data

Semester Genap 2021/2022

Outline

- Why MapReduce
- The MapReduce framework
- MapReduce Job
- Map and Reduce tasks
- Use case of MapReduce

Why MapReduce ?

- Distributes the processing of data on your cluster
- Divides your data up into partitions that are MAPPED (transformed) and REDUCED (aggregated) by mapper and reducer functions you define
- Resilient to failure – an application master monitors your mappers and reducers on each partition

The MapReduce framework

- MapReduce job types:
  - Single mapper jobs
  - Single mapper reducer jobs
  - Multiple mappers reducer jobs
- MapReduce patterns:
  - Aggregation patterns
  - Filtering patterns
  - Join patterns

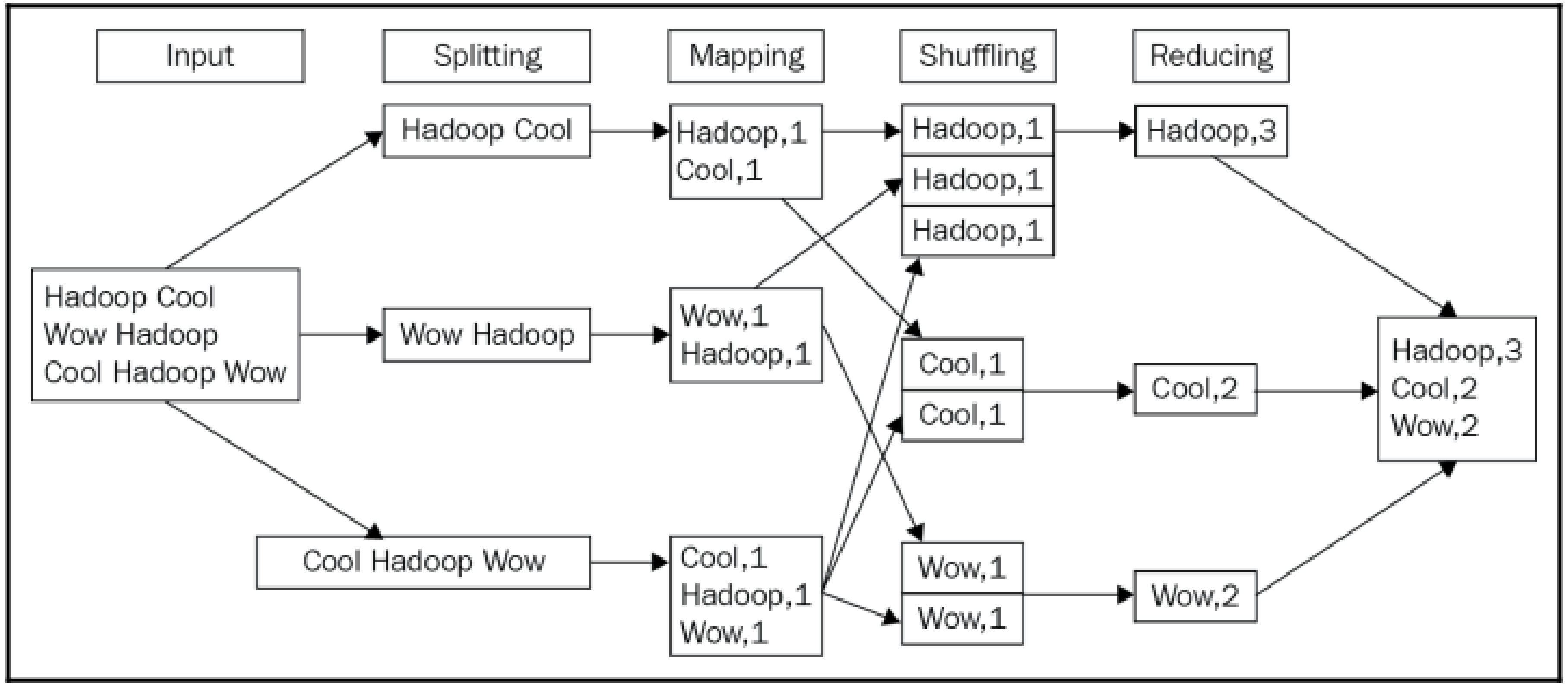
The MapReduce framework

- MapReduce is a framework used to compute a large amount of data in a Hadoop cluster.
- MapReduce uses YARN to schedule the mappers and reducers as tasks, using the containers.
- The MapReduce framework enables you to write distributed applications to process large amounts of data from a filesystem, such as a **Hadoop Distributed File System** (**HDFS**), in a reliable and fault-tolerant manner.

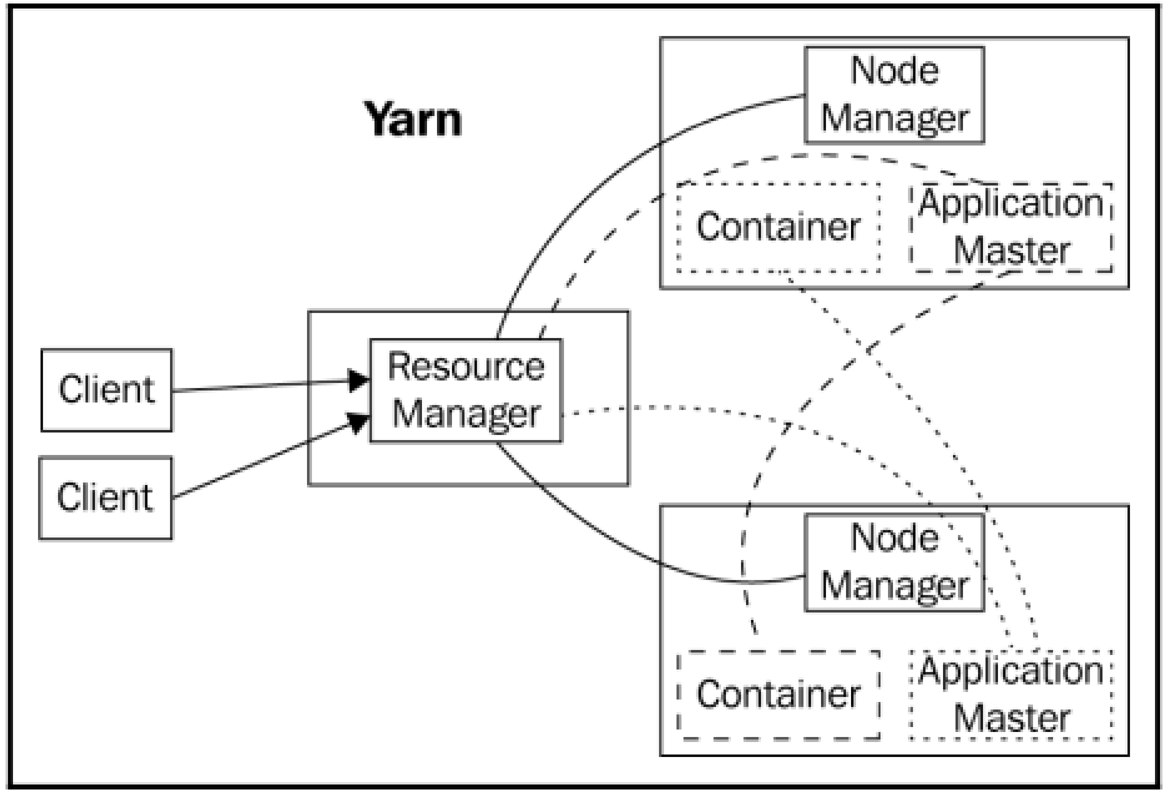
The MapReduce framework

- A MapReduce job usually works by splitting the input data across worker nodes, running the mapper tasks in a parallel manner.
- At this time, any failures that happen, either at the HDFS level or the failure of a mapper task, are handled automatically, to be fault-tolerant.
- Once the mappers have completed, in the results are copied over the network to other machines running the reducer tasks.

An example of using a MapReduce job to count frequencies of words is shown in the following diagram:



MapReduce uses YARN as a resource manager, which is shown in the following diagram:



The MapReduce framework

- The term MapReduce actually refers to **two separate** and distinct tasks that Hadoop programs perform.
- **The first is the map job**, which takes a set of data and converts it into another set of data, where individual elements are broken down into **tuples** (**key/value pairs**).
- **The second is reduce job**, takes the output from a map as input and combines those data tuples into a **smaller set of tuples**.
- As the sequence of the name MapReduce implies, **the reduce job is always performed after the map job**.

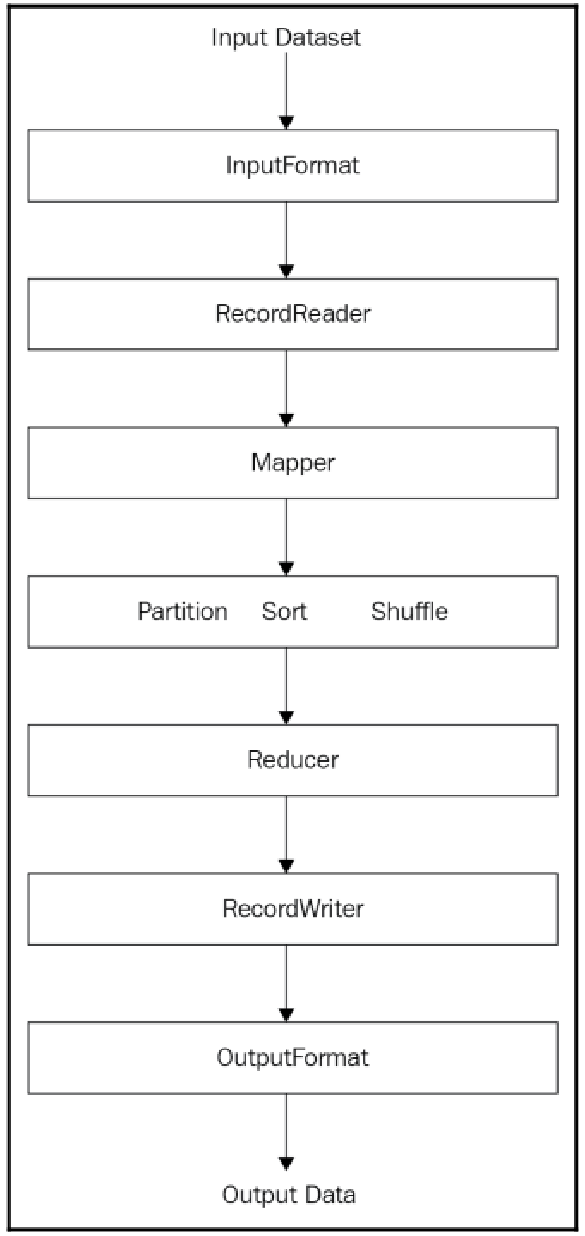
MapReduce Job

- The input to a MapReduce job is a set of files in the data store that is spread out over the HDFS.
- In Hadoop, these files are split with an input format, which defines how to separate a file into input splits.
- An input split is a byte-oriented view of a chunk of the file, to be loaded by a map task.
- Each **map task** in Hadoop is broken into the following phases:
  - record reader,
  - mapper,
  - combiner, and
  - partitioner.

MapReduce Job

- The output of the map tasks, called the **intermediate keys and values**, is sent to the reducers.
- The **reduce tasks** are broken into the following phases:
  - shuffle,
  - sort,
  - reducer, and
  - output format.
- The nodes in which the map tasks run are optimally on the nodes in which the data rests. This way, the data typically does not have to move over the network, and can be computed on the local machine.

The Programming Components of a MapReduce Program



Map Task: 1. Record Reader

- The input reader divides the input into appropriately sized splits (in practice, typically, **64 MB to 128 MB**), and the framework assigns one split to each map function.
- The input reader reads data from stable storage (typically, a distributed filesystem) and generates key/value pairs.
- The record reader **translates** an input split generated by input format into records.
- The **purpose of the record reader is to parse the data into records**, but not to parse the record itself.
- It passes the data to the mapper in the **form of a key/value pair**.

Map Task: 2. Mapper

- The `map` function takes a series of key/value pairs, processes each, and generates zero or more output key/value pairs.
- The input and output types of the map can be (and often are) different from each other.
- In the mapper, code is executed on each key/value pair from the record reader to produce zero or more new key/value pairs, called the **intermediate output of the mapper** (which also consists of key/value pairs).
- The **key** is what the data will be grouped on and the **value** is the part of the data to be used in the reducer to generate the necessary output.

Map Task: 3. Combiner

- If every output of every mapper is directly sent over to every reducer, this will consume a significant amount of resources and time.
- The combiner, an **optional localized reducer**, can group data in the map phase.
- It takes the intermediate keys from the mapper and applies a user-provided method to aggregate values in the small scope of that one mapper.
- We will point out which patterns benefit from using a combiner, and which ones cannot use a combiner.
- A **combiner is not guaranteed to execute**, so it cannot be a part of the overall algorithm.

Map Task: 4. Partitioner

- The partitioner takes the intermediate key/value pairs from the mapper (or combiner if it is being used) and splits them up into shards, one shard per reducer.
- Each `map` function output is allocated to a particular reducer by the application's `partition` function for sharding purposes.
- The `partition` function, is given the key and the number of reducers and returns the index of the desired reducer.
- A typical default is to hash the key and use the `hash` value to module the number of reducers:

```
partitionId = hash(key) % R, where R is number of Reducers
```

Reduce Task: 1. Shuffle and Sort

- Once the mappers are done with the input data processing (essentially, splitting the data and generating key/value pairs), the output has to be distributed across the cluster to start the reduce tasks.
- Hence, a reduce task starts with the shuffle and sort step, by taking the output files written by all of the mappers and subsequent partitioners and downloads them to the local machine in which the reducer task is running.
- These individual data pieces are then sorted by key into one larger list of key/value pairs. The **purpose of this sort is to group equivalent keys together**, so that their values can be iterated over easily in the reduce task.

Reduce Task: 2. Reducer

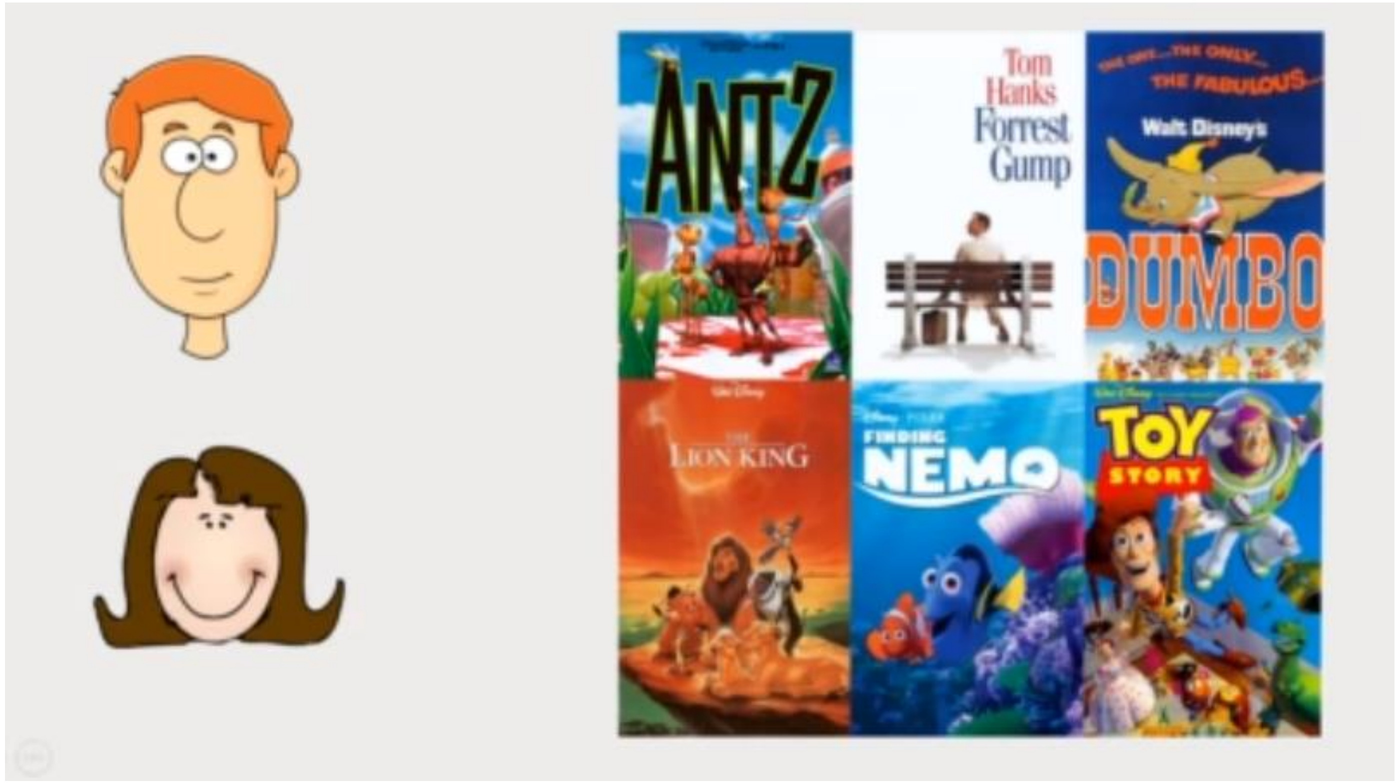
- The reducer takes the grouped data as input and runs a `reduce` function once per key grouping.
- The function is passed the key and an iterator over all of the values associated with that key.
- The **data can be aggregated, filtered, and combined** in a number of ways. Once the reduce function is done, it sends zero or more key/value pairs to the final step, the output format.

Reduce Task: 3. Output Format

- The output format translates the final key/value pair from the `reduce` function and writes it out to a file by a record writer.
- By default, it will separate the key and value with a tab and separate records with a newline character.
- This can typically be **customized to provide richer output formats**, but in the end, the **data is written out to HDFS**, regardless of format.
- Not only is writing to HDFS supported by default but also output to Elasticsearch index, output to RDBMS, or a NoSQL such as Cassandra, HBase, and so on.

Let’s illustrate with an example

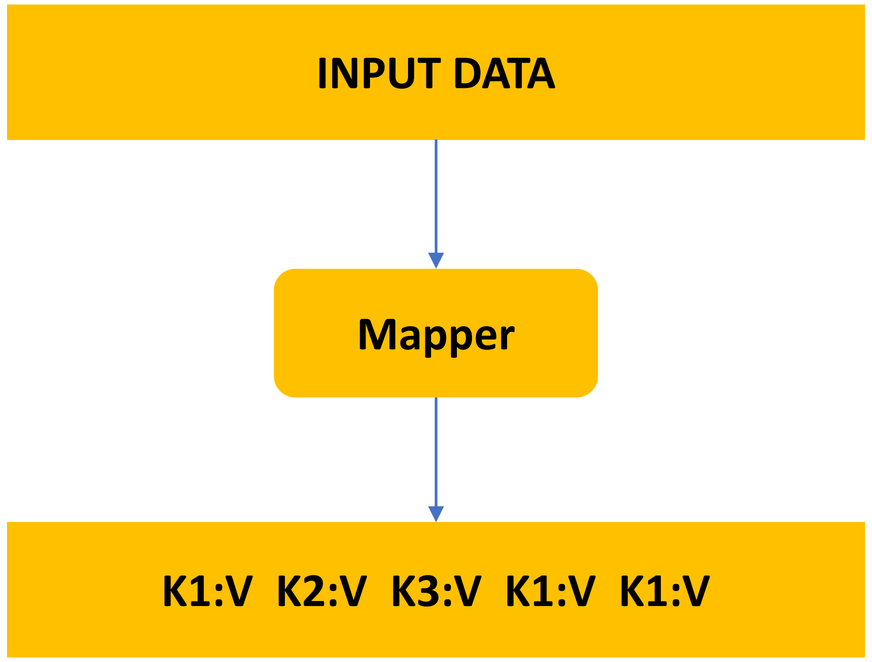
How many movies did each user rate in the MovieLens data set ?



https://grouplens.org/datasets/movielens/

How MapReduce works: Mapping

- The **MAPPER** converts raw source data into **key/value** pairs



Example: MovieLens Data

https://grouplens.org/datasets/movielens/

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182906
186	265	2	881171488

Map users to movies they watched

https://grouplens.org/datasets/movielens/

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182906
186	265	2	881171488

$$\begin{tikzcd} \text{} \arrow[d] \\ \textbf{Mapper} \arrow[d] \\ \text{} \end{tikzcd}$$

196:242 186:302 196:377 244:51 166:346 186:474 186:265

Extract and Organize What We Care About

196:242 186:302 196:377 244:51 166:346 186:474 186:265

MapReduce Sorts and Groups the Mapped Data ("Shuffle and Sort")

196:242 186:302 196:377 244:51 166:346 186:474 186:265

$$\downarrow$$

166:346   186:302,474,265   196:242,377   244:51

The REDUCER Processes Each Key’s Values

$$\begin{tikzcd}
\text{166:346 \quad 186:302,474,265 \quad 196:242,377 \quad 244:51} \arrow[d] \\
\text{len(movies)} \arrow[d] \\
\text{166:1 \quad 186:3 \quad 196:2 \quad 244:1}
\end{tikzcd}$$

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182906
186	265	2	881171488

Mapper

196:242 186:302 196:377 244:51 166:346 186:474 186:265

SHUFFLE AND SORT

166:346 186:302,474,265 196:242,377 244:51

REDUCER

166:1 186:3 196:2 244:1

Tugas

- Jelaskan jenis-jenis *job* dan *pattern* MapReduce pada slide no. 4 beserta skenario penggunaannya!

Referensi

- https://www.udemy.com/course/the-ultimate-hands-on-hadoop-tame-your-big-data/learn/lecture/5951208
- Big Data Analytics with Hadoop 3 (2018)