MAP AND REDUCE

In Spark

Big Data H/M 2022

Richard McCreadie

A QUICK REMINDER ON THE ORIGIN OF MAP-REDUCE

- Map-Reduce was originally proposed as a processing paradigm by Google in 2004

Original Print

- It was one of their solutions for how to manage the distributed processing of very large datasets (such as their web crawls)

GOOGLE'S MAP-REDUCE

- The original Map-Reduce paper was concerned with what an interface/API for general massively parallel distributed computation might look like
- The first main contribution of the Map-Reduce paper was the specification of two API methods, with an intermediate shuffle step:

```
map (k1, v1) → list(k2, v2)
shuffle (sort and group)
reduce (k2, list(v2)) → list(v2)
```

- These are transformation functions
  - Map takes a <key,value> pair, performs some processing on it, and then outputs 0 or more <key,value> pairs, which can be of a different type to the input
  - The shuffle step groups map output by key
  - Reduce takes the shuffle output (batch of map output for a key) and then processes it, producing a (smaller) list of items of the same type

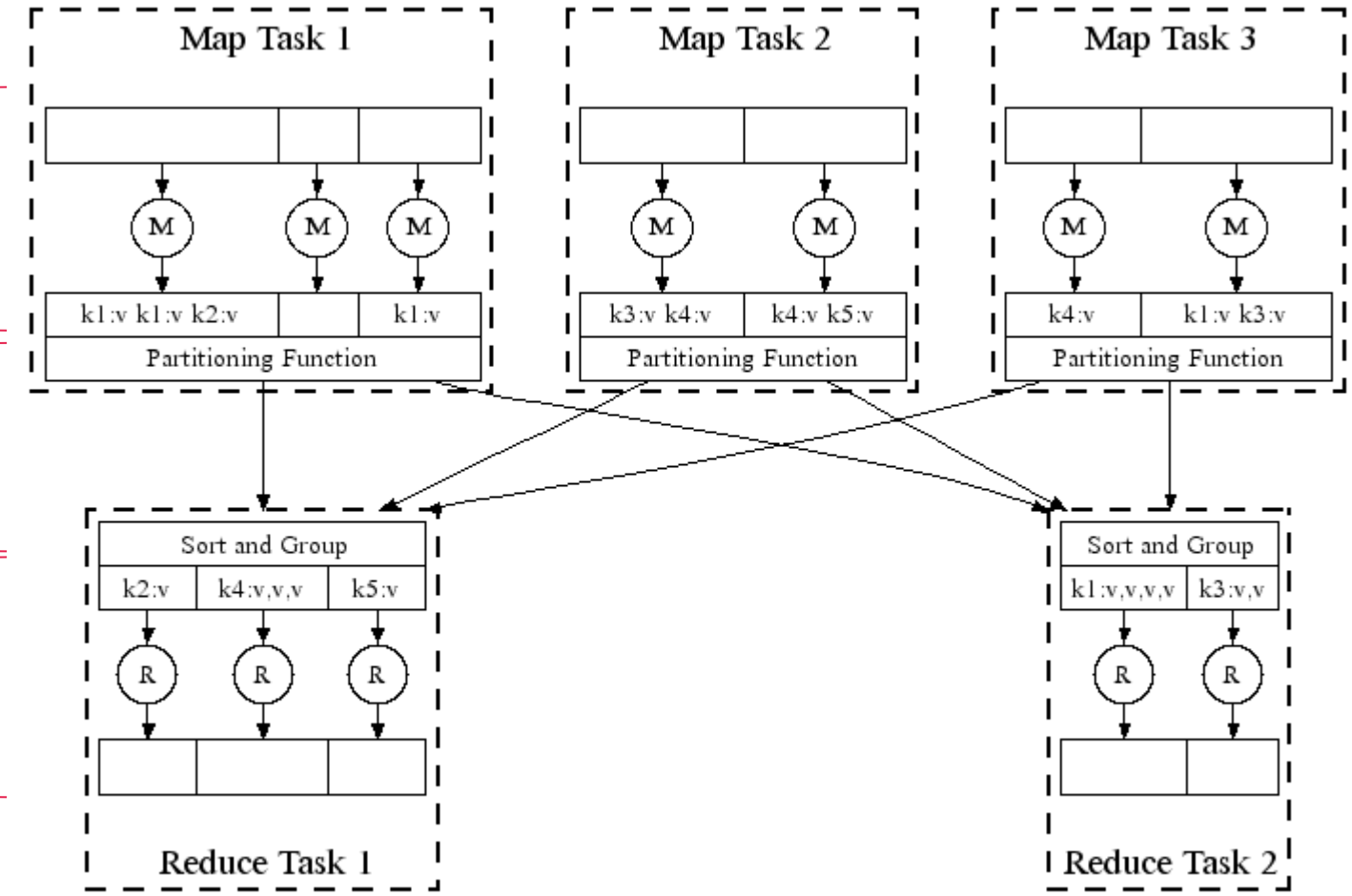
VISUALISATION

`map (k1, v1) → list(k2, v2)`

`shuffle (sort and group)`

(This involved each map task writing to a shared data store, then each reducer reading from it, memory was small and expensive at this time)

`reduce (k2, list(v2)) → list(v2)`



https://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0008.html

SPARK WORKS DIFFERENTLY!

SPARK DATA STRUCTURES: RDD'S

- Google's Map-Reduce paradigm defined computation in terms of lists of key,value pairs
- Spark's developers noted that not all transformations needed to have explicit keys and instead defined computation in terms of collections of data tuples, known as *resilient distributed datasets* (RDDs)
  - Tuples could be of any length and contain any types (that can be serialized)
  - The are schema-less (there is no specification of what should be in each tuple)

<1,"john", "smith", "MSc", 3.12>
<2,"xi", "wang", "MSc", 3.25>
<3,"jin", "lin", "MSc", 3.11>

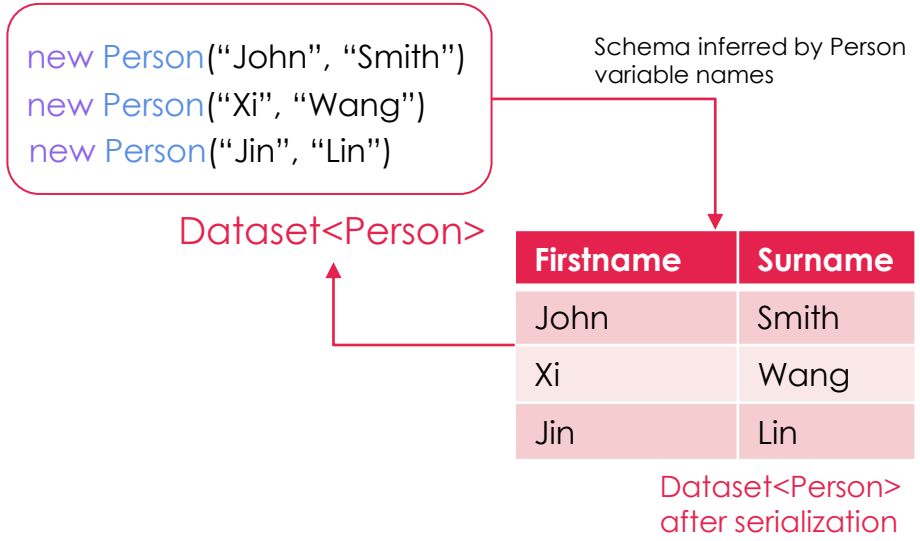
RDD

SPARK DATA STRUCTURES: DATAFRAMES AND DATASETS

- Spark SQL was a later addition, which introduced an alterative data structure called a DataFrame
  - This can be thought of simply as an RDD with a schema, like a table header
  - They were added to make doing SQL operations in Spark easier
- Datasets were added in 2016, and act as an abstraction for both RDDs and DataFrames
  - A Dataset<Row> == A Dataframe
  - A Dataset<AnythingElse> introduces strong type checking

Id	Firstname	Surname	Degree	GPA
1	John	Smith	MSc	3.12
2	Xi	Wang	MSc	3.25
3	Jin	Lin	MSc	3.11

DataFrame



SPARK MAP AND REDUCE

- Apache Spark also does NOT use the same definition of map and reduce as in Google's original paper
  - Spark was released 10 years later
  - The hardware landscape had changed, memory was much cheaper
  - Having only linear sequences of map->reduce operations was seen as too restrictive

- Instead, Spark defines a wide range of transformation functions

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numPartitions*]))	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numPartitions*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. **Note:** If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using `reduceByKey` or `aggregateByKey` will yield much better performance.

…and many more

https://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

MAP AND FLATMAP IN SPARK

- The map operator in spark can be thought of a distributed way of performing a 'foreach' on each item in the Dataset (or RDD or Dataframe)
  - Converts each item in the dataset from one type to another

```
map (v1) → (v2)
```

- Spark then introduces a second flatmap function that acts more like Google's version of map
  - Instead of performing a 1-1 mapping, it performs a 1-many mapping

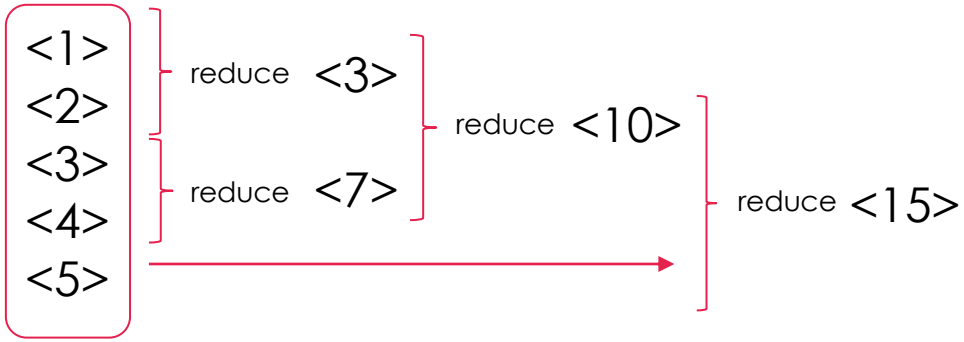
```
flatmap (v1) → Iterator<v2>
```

This iterator may output any number of items, including 0 if you need to filter your dataset

REDUCE IN SPARK

- Unlike in Google's design, the reduce operator in Spark defines computation in a pair-wise fashion instead of merging a list
  - A reduce operation merges two items of the same type and outputs a new item of that same type

```
reduce (v1, v2) → (v3)
```



SUM Example

- Reduce will be called iteratively for each pair in the Dataset until a single output value is obtained.
  - A classic example of a reduce function might be to sum values together