DDAM, 2019
HADOOP COMBINER

Docente: Patrizio Dazzi

Email: patrizio.dazzi@isti.cnr.it

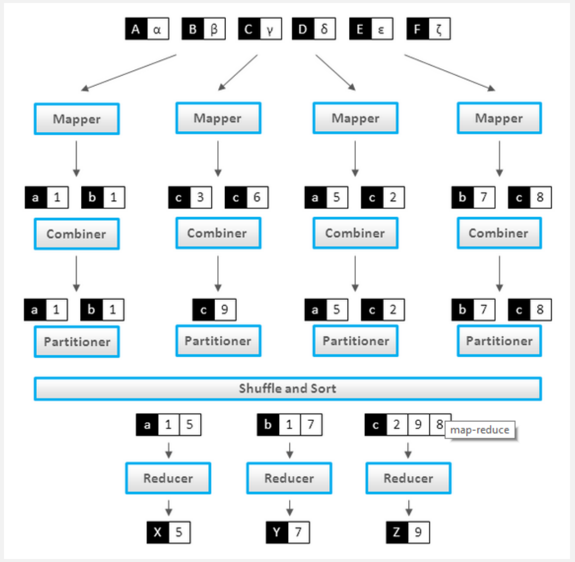
A DESIGN PATTERN

A design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

MAP REDUCE++

- The **Combiner** class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.
  Default: merge same keys locally.

- A **Partitioner** partitions the key-value pairs of intermediate Map-outputs. It partitions the data using a user-defined condition, which works like a hash function. The total number of partitions is same as the number of Reducer tasks for the job.
  Default: each key generates a reducer.



COUNTING AND SUMMING

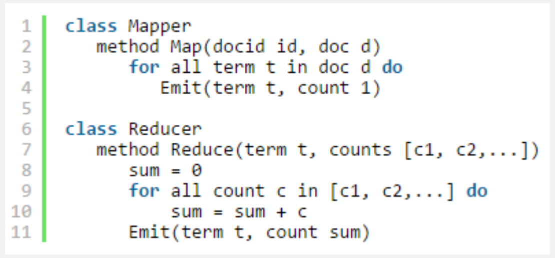
Problem Statement: There is a number of documents where each document is a set of terms. It is required to calculate a total number of occurrences of each term in all documents. Alternatively, it can be an arbitrary function of the terms. For instance, there is a log file where each record contains a response time and it is required to calculate an average response time.

Applications:

- Log Analysis, Data Querying

COUNTING AND SUMMING (SOLUTION)

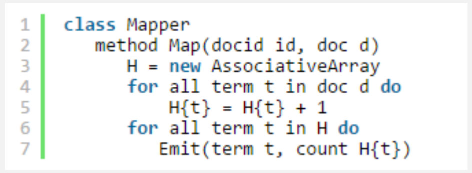
- Let start with something really simple. The code snippet below shows Mapper that simply emit "1" for each term it processes and Reducer that goes through the lists of ones and sum them up:



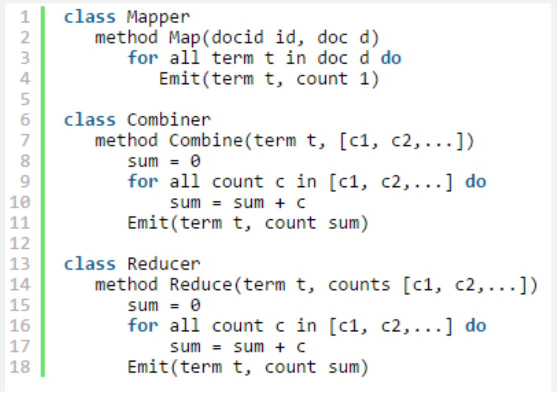
```
class Mapper
   method Map(docid id, doc d)
      for all term t in doc d do
         Emit(term t, count 1)

class Reducer
   method Reduce(term t, counts [c1, c2,...])
      sum = 0
      for all count c in [c1, c2,...] do
          sum = sum + c
      Emit(term t, count sum)
```

- The obvious disadvantage of this approach is a high amount of dummy counters emitted by the Mapper. The Mapper can decrease a number of counters via summing counters for each document:



```
class Mapper
   method Map(docid id, doc d)
      H = new AssociativeArray
      for all term t in doc d do
          H{t} = H{t} + 1
      for all term t in H do
         Emit(term t, count H{t})
```



```
class Mapper
   method Map(docid id, doc d)
      for all term t in doc d do
         Emit(term t, count 1)

class Combiner
   method Combine(term t, [c1, c2,...])
      sum = 0
      for all count c in [c1, c2,...] do
          sum = sum + c
      Emit(term t, count sum)

class Reducer
   method Reduce(term t, counts [c1, c2,...])
      sum = 0
      for all count c in [c1, c2,...] do
          sum = sum + c
      Emit(term t, count sum)
```