Lab 3 – Alternative solutions

Lab 3

- Input file

A1008ULQSWI006,**B**0017OAQIY
A100EBHBG1GF5,B0013T5YO4
A1017Y0SGBINVS,B0009F3SAK
A101F8M8DPFOM9,B005HY2BRO,B000H7MFVI
A102H88HCCJJAB,B0007A8XV6
A102ME7M2YW2P5,B000FKGT8W
A102QP2OSXRVH,B001EQ5SGU,B000EH0RTS
A102TGNH1D915Z,B000RHXKC6,B0002DHNXC,B0002DHNXC,B000XJK7UG,B00008DFK5,B000SP1CWW,B0009YD7P2,B000SP1CWW,B00008DFK5,B0009YD7P2
A1051WAJL0HJWH,B000W5U5H6
A1052V04GOA7RV,B002GJ9JY6,B001E5E3JY,B008ZRKZSM,B002GJ9JWS
....

- Each line contains
  - a reviewer ID (**A**XXXXXX) and
  - the list of products reviewed by her/him (**B**XXXXXX)

Lab 3

- Your goal is to find the top 100 pairs of products most often reviewed (and so bought) together
- We consider two products as reviewed (i.e., bought) together if they appear in the same line of the input file

Lab 3: Possible solutions

- At least three different “approaches” can be used to solve Lab 3

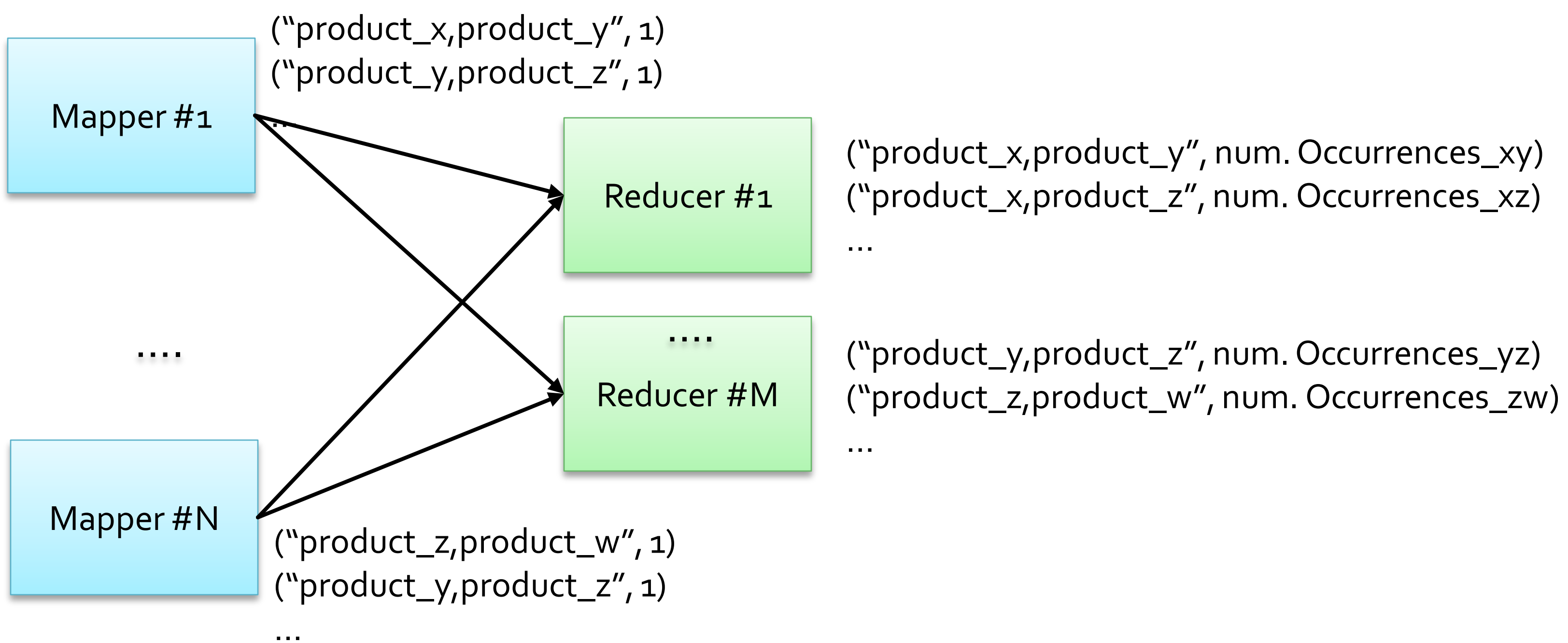
Solution #1

Lab 3: Solution #1

1. A chain of **two** MapReduce **jobs** is used
   - The **first job computes the number of occurrences of each pair of products** that occur together in at least one line of the input file
     - It is like a word count where each "word" is a pair of products
   - The **second job selects the top-k pairs** of products, in terms of num. of occurrences, among the pairs emitted by the first job
     - It implements the top-k pattern

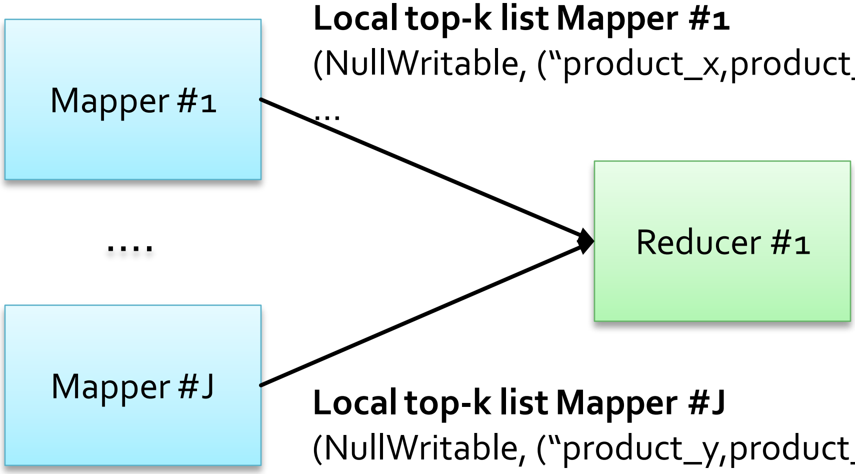
Lab 3: Solution #1

- The first job computes the number of occurrences of each pair of products analyzing the input file



Lab 3: Solution #1

- The second job computes the global top-k pairs of products in terms of num. of occurrences



Local top-k list Mapper #1
(NullWritable, (“product_x,product_y”, num. Occurrences_xy))
…

Global top-k list
(“product_x,product_y”, num. Occurrences_xy)
(“product_y,product_z”, num. Occurrences_yz)
…

Local top-k list Mapper #J
(NullWritable, (“product_y,product_z”, num. Occurrences_yz))
…

Solution #2

Lab 3: Solution #2

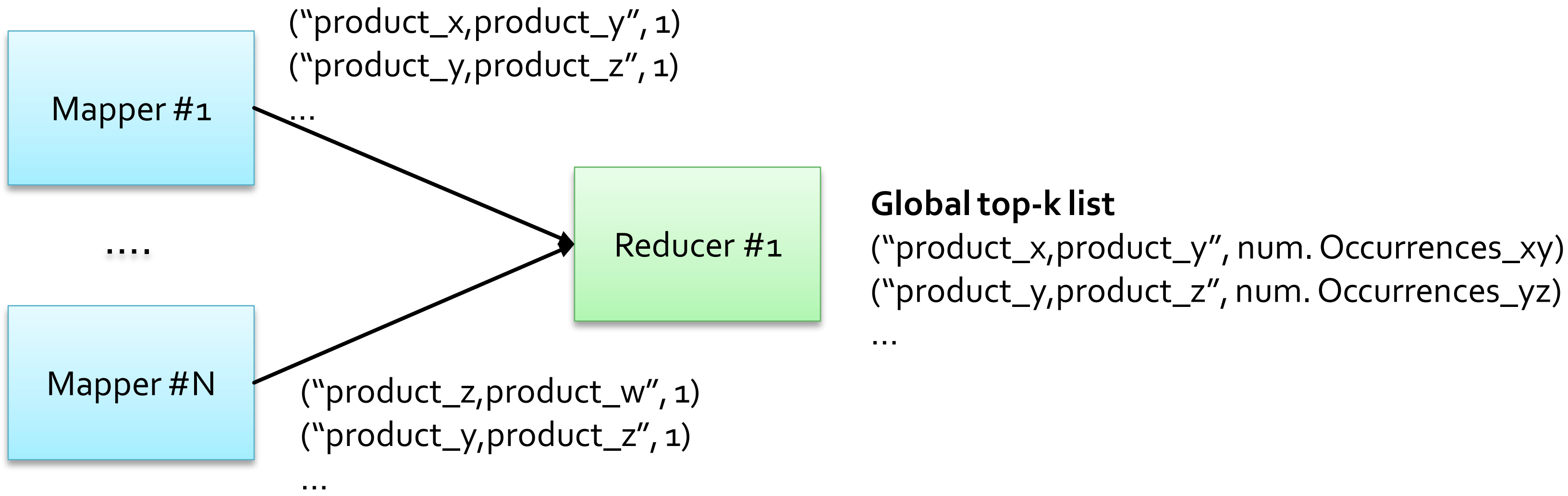
2. **One single** MapReduce **job** is used
   - The job
     - **Computes the number of occurrences of each pair of products**
       - It is again like a word count where each "word" is a pair of products
       - However, the reducer does not emit all the pairs (pair of products, #of occurrences) that it computes
       - The top-k list is computed in the reducer and is emitted in its cleanup method

Lab 3: Solution #2

- In the **reducer**, the job **computes** **also** **the top-k list**
  - By initializing the top-k list in the setup method of the reducer
  - By updating the top-k list in the reduce method (immediately after the computation of the frequency of the current pair of products)
  - By emitting the final top-k list in the cleanup method of the reducer
- There must be **one single instance of the reducer** in order to compute the final global top-k list

Lab 3: Solution #2

- There is one single job that computes the number of occurrences and the global top-k list at the same time in its single instance of the reducer



Solution #3

Lab 3: Solution #3

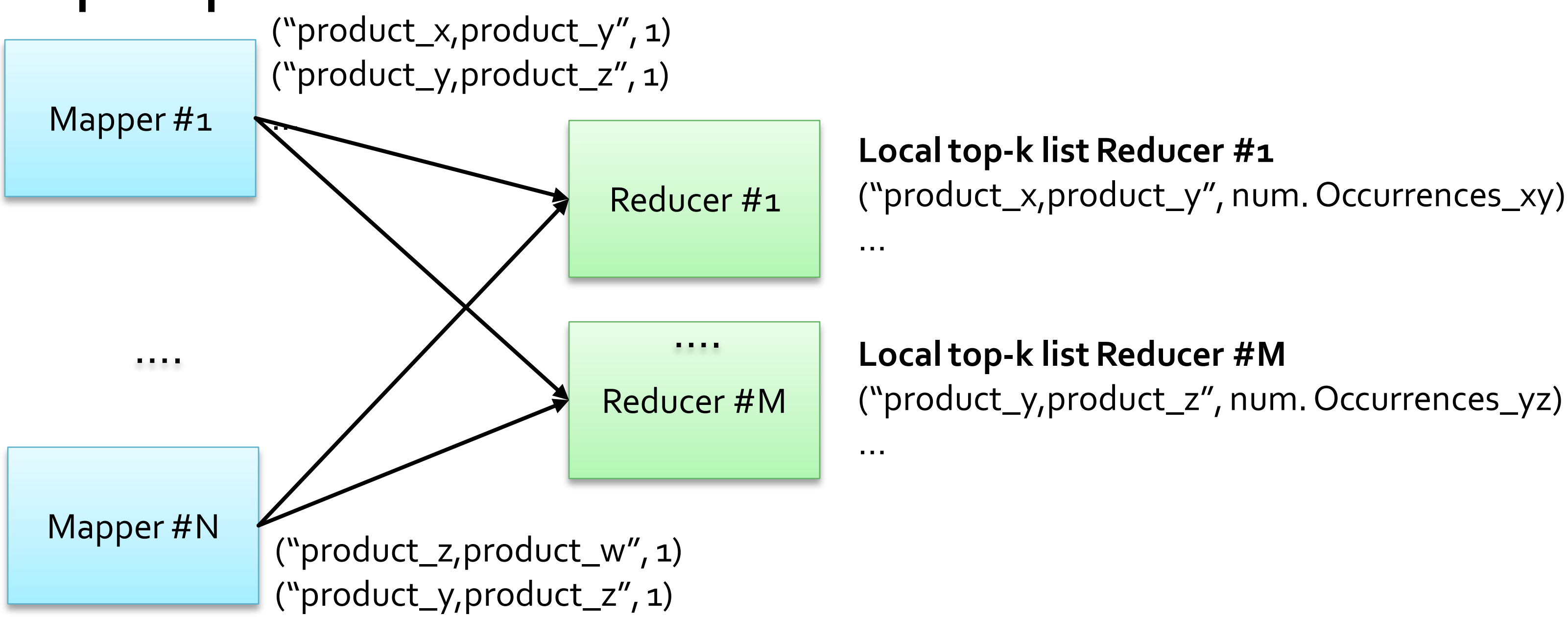
3. A chain of **two** MapReduce **jobs** is used
   - The first job is the same job used by Solution #2
     - However, in this case the number of instances of the reducers class is set to a value greater than one
       - This setting allows parallelizing the reduce step of the first job
     - **Each reducer emits a local top-k list**
       - The first job returns a number of local top-k lists equal to the number of reducers of the first job

Lab 3: Solution #3

- The **second job computes the final top-k** list merging the pairs of the local top-k lists emitted by the first job
  - It is based on the standard Top-k pattern

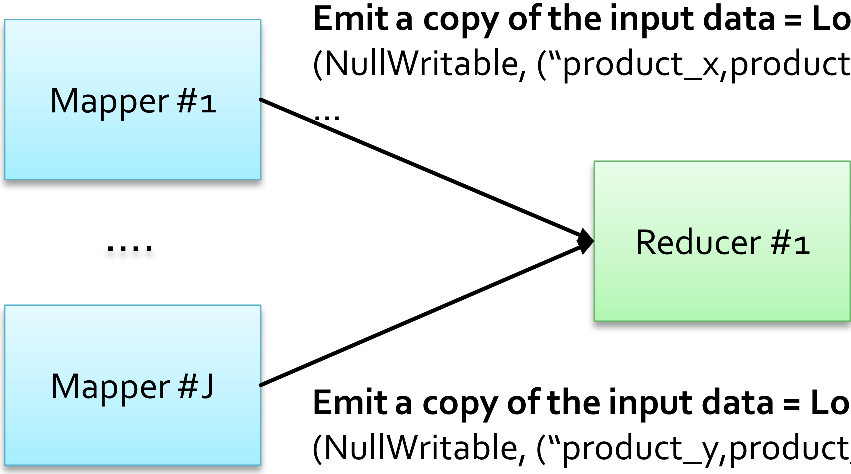
Lab 3: Solution #3

- The first job computes the number of occurrences of each pair of products but each instance of the reducer emits only its local top-k pairs



Lab 3: Solution #3

- The second job computes the global top-k pairs of products in terms of num. of occurrences merging the local list of job #1



Emit a copy of the input data = Local top-k list emitted by a reducer of Job #1
(NullWritable, ("product_x,product_y", num. Occurrences_xy))
...

Global top-k list
("product_x,product_y", num. Occurrences_xy)
("product_y,product_z", num. Occurrences_yz)
...

Emit a copy of the input data = Local top-k list emitted by a reducer of Job #J
(NullWritable, ("product_y,product_z", num. Occurrences_yz))
...

Comparison of the three proposed solutions

Lab 3: Comparison of the proposed solutions

- Solution #1
  - + Adopts two standard patterns
  - - However, the output of the first job is very large
    - One pair for each pair of products occurring together at least one time in the input file

Lab 3: Comparison of the proposed solutions

- Solution #2
  - +Only one job is instantiated and executed (there is only one job in Solution #2) and its output is already the final top-k list
  - - However, only one reducer is instantiated
    - It could become a bottleneck because one single reducer must analyze the potentially large set of pairs emitted by the mappers sequentially
    - The **slowest** of the three solutions
  - **This solution should not be used**
    - It is **highly inefficient**

Lab 3: Comparison of the proposed solutions

- Solution #3
  - +Each reducer of the first job emits only the pair contained in its local top-k lists
    - One top-k list for each reducer
    - The pairs of the top-k lists emitted by the reducers are significantly smaller than all the pairs of products occurring together at least one time
    - Since the first job instantiates many reducers, the parallelism is maintained for the first job that is the heaviest one
  - \- It is not a standard pattern