

# **Corso di BigData**

## **Primo Progetto**

**26 aprile 2018**

**Nome gruppo:** **RD Data**

**Componenti:** **Ronchetti Claudio, De Matteis Valerio**

## **Descrizione del progetto**

Progettare e realizzare in: (a) MapReduce, (b) Hive e (c) Spark:

1. Un job che sia in grado di generare, per ciascun anno, le dieci parole che sono state più usate nelle recensioni (campo summary) in ordine di frequenza, indicando, per ogni parola, la sua frequenza, ovvero il numero di occorrenze della parola nelle recensioni di quell'anno.
  
2. Un job che sia in grado di generare, per ciascun prodotto, lo score medio ottenuto in ciascuno degli anni compresi tra il 2003 e il 2012, indicando ProductId seguito da tutti gli score medi ottenuti negli anni dell'intervallo. Il risultato deve essere ordinato in base al ProductId.
  
3. Un job in grado di generare coppie di prodotti che hanno almeno un utente in comune, ovvero che sono stati recensiti da uno stesso utente, indicando, per ciascuna coppia, il numero di utenti in comune. Il risultato deve essere ordinato in base allo ProductId del primo elemento della coppia e, possibilmente, non deve presentare duplicati.

## PSEUDOCODICE

# MAP REDUCE

## JOB1:

**MAP( value ) :**

```
rowFields = value.split(',')
```

```
year = rowFields[TIME].getYear
```

*/get year from TIME field*

```
summary = rowFields[SUMMARY]
```

*/get summary from SUMMARY*

## *field*

`cleanLine = CLEAR(summary)  
useless symbols`

*/clear the text in summary from*

```
tokens = TOKENIZE(cleanLine)  
text
```

*/transform the text in a list of*

for token in tokens :

**WRITE( year , token )**

/write year-token pair in context

**REDUCE(key , values, countOccurrenceMap)**

**if not countOccurrenceMap.contains(key) then**

```
mapTmp = {}
```

else

```
mapTmp = countOccurrenceMap.getValue(key)
```

for word in values :

```
mapTmp.getValue(word) ++
```

*/update the occurrence of*

word

```
countOccurrenceMap.put( key , mapTmp)
```

**CLEANUP(countOccurrenceMap)**

```
for year in countOccu  
    ordered = countOc
```

**WRITE ( year , TOP10(ordered) )**

## JOB2:

**MAP( value ) :**

```
rowFields = value.split(',')
year = rowFields[TIME].getYear                                /get year from TIME field
prodID = rowFields[PRODUCTID]                                 /get prodID from
PRODUCTID field
score = rowFields[SCORE]                                     /get score from SCORE
eld
if 2003 <= year <= 2012 then
    mapTmp = scoreMap.getvalue( prodID )                     /if don't exist mapTmp is NULL
    listTmp = mapTmp.getValue(year)                           /if don't exist listTmp is NULL
    listTmp.addValue(score)
```

**CLEANUP(context, scoreMap)**

```
for prodID in scoreMap.keySet  
    for year in scoreMap[prodID]  
        length = scoreMap[ProdID][year].length  
        count = scoreMap[ProdID][year].count()  
        WRITE (prodID, ( year , length, count ))  
/write the triple in
```

**REDUCE(key, values, scoreMap)**

```
for (year , length , count) in values :  
    tuple = scoreMap [key]  
    for (year_t , length_t , count_t) in tuple:  
        if (year_t == year) then  
            scoreMap [key] = (year_t , length_t + length, count_t + count)  
            /update  
            example of key  
            break  
            /and exit
```

#### CLEANUP(scoreMap)

```
for prodID in scoreMap.keySet() :  
    scoreMap[prodID].ORDERbyYear()  
    averageList = []  
    for (year , length , count) in scoreMap[prodID] :  
        averageList += [count/length]  
    WRITE( prodID , averageList )
```

## JOB3:

MAP(value) :

```
rowFields = value.split(',')
prodID = rowFields[PRODUCTID]           /get prodID from PRODUCTID field
userID = rowFields[USERID]              /get userID from USERID field
WRITE(userID,prodID)
```

REDUCE(key, values, productsCouples)

```
productList = []
for prodID in values :
    productList += [prodID]
for prod_A in productList :
    for prod_B in productList :
        if prod_A != prod_B then
            key = prod_A + prod_B           /key is the combination of
two prodID
            productsCouples[key] += 1       /update the count of couple,
```

CLEANUP(productsCouples)

```
for products in productsCouples :
    WRITE( products , productsCouples[products])
```

# SPARK

## JOB1:

```
yearWordCoupleCouple = context.textFile(pathToFile)
    .mapToPair( lambda: line => (year, summary))
    .flatMapValues( lambda : summary => summary.split(" "))
    .mapToPair( lambda : (year,word) => ((year, word), 1))

orderedResult= yearWordCouple.reduceByKey( lambda : (_, _) => _ + _)
    .map( lambda : ((year, word) , count) => (year, word, count))
    .mapToPair( lambda: (year, word, freq) => ((year, count), word))
    .sortByKey( _.sortByYear().sortByFreq())

resultOrderedTriple = orderedResult.map( lambda : ((year, count), word) => (year, count,
word) )

result = resultOrderedTriple.groupByKey()
    .sortByKey()
return result
```

## JOB2:

```
yearsProductScore = context.textFile(pathToFile)
    .mapToPair( lambda : line => ((prodID, year), score))
    .filter( lambda : year => (2013 > year > 2002))

prodAveragesList= yearsProductScore.aggregateByKey()
    .sortByKey(sortByYear, sortByProductID)
    .map( lambda : ((prodID, year), avg) => (prodID, year, avg) );

result = prodAveragesList.groupByKeyProductID()
    .sortByKey()
return result
```

## JOB3:

```
userProd= context.textFile(pathToFile)
    .mapToPair( lambda : line => (userID, prodID))
aggregated= userProd.groupByKey()
userProdJoined = aggregate.join(aggregate)

coupleProdUser = userProdJoined.flatMapToPair( lambda : _ => (prodA, prodB, n))
    / dove prodA e prodB sono la coppia di prodotti
acquistati da n persone

result = coupleProdUser.reduceByKey( lambda : (_,_) => _ + _)
    .sortByKey(sortByFirstProductId, sortBySecondProductId)
return result
```

# RISULTATI

## Risultati JOB1:

1999	a: 3
1999	day: 3
1999	fairy: 3
1999	modern: 3
1999	tale: 3
1999	is: 2
1999	book: 1
1999	child: 1
1999	educational: 1
1999	entertainingl: 1
2000	a: 11
2000	version: 5

...

2011	not: 8360
2011	love: 8021
2012	great: 23137
2012	good: 16889
2012	the: 16357
2012	for: 12417
2012	a: 12354
2012	and: 10498
2012	not: 10066
2012	love: 9678
2012	coffee: 9621
2012	my: 9601

## Risultati JOB2:

NOTA: con il simbolo ---- si rappresenta l'assenza di uno score del prodotto in quello specifico anno

0006641040 [5.00, 4.33, 3.25, ----, 4.50, 4.00, 5.00, 5.00, 4.16, 4.00]  
141278509X [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
2734888454 [----, ----, ----, ----, 3.50, ----, ----, ----, ----, ----]  
2841233731 [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
7310172001 [----, ----, 3.50, 5.00, 4.90, 4.54, 4.69, 4.77, 4.78, 4.79]  
7310172101 [----, ----, 3.50, 5.00, 4.90, 4.54, 4.69, 4.77, 4.78, 4.79]  
7800648702 [----, ----, ----, ----, ----, ----, ----, ----, ----, 4.00]  
9376674501 [----, ----, ----, ----, ----, ----, ----, 5.00, ----]  
B00002N8SM [----, ----, ----, ----, 2.00, 1.00, 2.50, 1.25, 2.25, 1.55]  
B00002NCJC [----, ----, ----, ----, ----, ----, 4.50, ----, ----]

...

B009RSR8HO [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
B009SA5NNW[----, ----, ----, ----, ----, ----, ----, ----, 3.50, 4.00]  
B009SF0TN6 [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
B009SMKESO [----, ----, ----, ----, ----, ----, ----, ----, 4.00, ----]  
B009SR4OQ2 [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
B009UOFTUI [----, ----, ----, ----, ----, ----, ----, ----, ----, 1.00]  
B009UOFU20 [----, ----, ----, ----, ----, ----, ----, ----, ----, 1.00]  
B009UUUS05I [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
B009WSNWC4 [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]  
B009WVB40S [----, ----, ----, ----, ----, ----, ----, ----, ----, 5.00]

## Risultati JOB3:

0006641040,B0005XN9HI	1
0006641040,B00061EPKE	1
0006641040,B000EM00YU	1
0006641040,B000FDQV46	1
0006641040,B000FV8LPU	1
0006641040,B000MGOZEO	1
0006641040,B000MLHU3M	1
0006641040,B000UVW59S	1
0006641040,B000UVZRES	1
0006641040,B000UW1Q8I	1

...

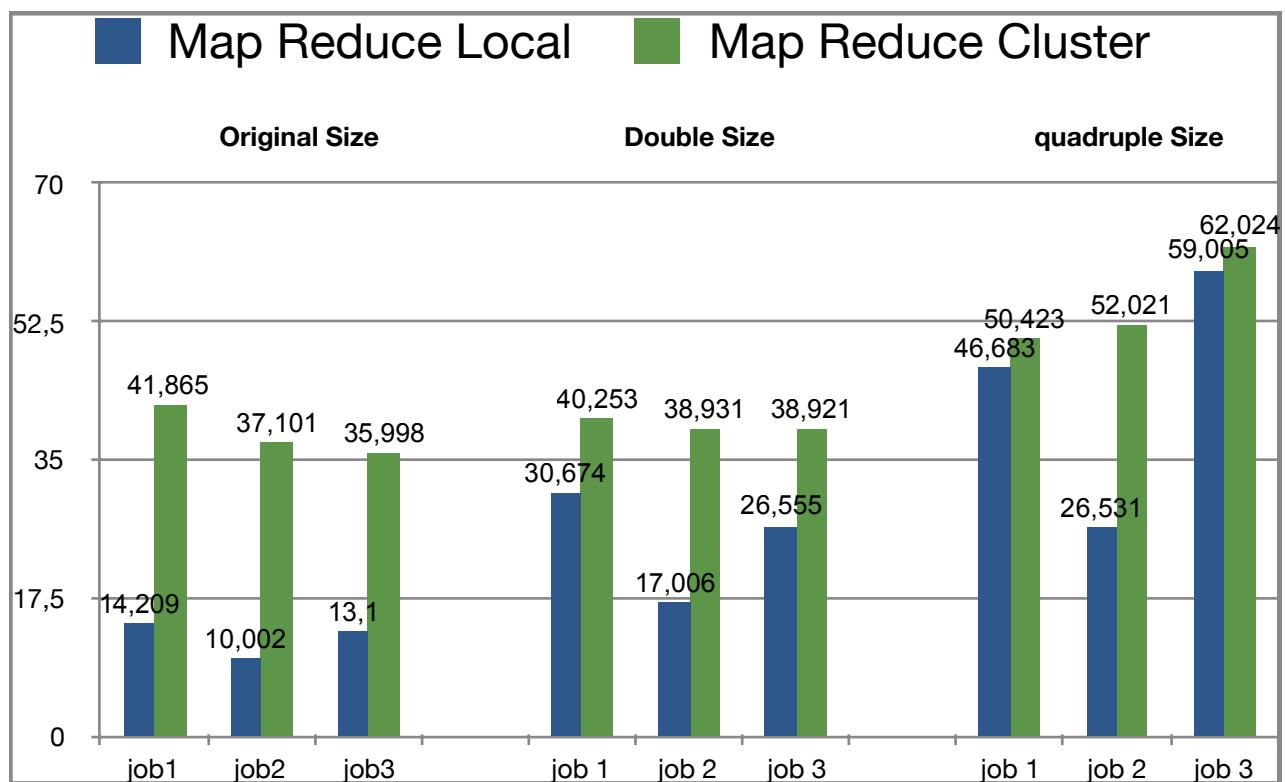
B009O753WA,B009O7DGEW	1
B009O753WA,B009PFJUF2	1
B009O753WA,B009PG8MVO	1
B009O7DGEW,B009PFJUF2	1
B009O7DGEW,B009PG8MVO	1
B009OM65GI,B009OM65H2	2
B009OM65GI,B009OM66IU	2
B009OM65H2,B009OM66IU	2
B009PFJUF2,B009PG8MVO	1
B009UOFTUI,B009UOFU20	1

# TEMPI DI ESECUZIONE

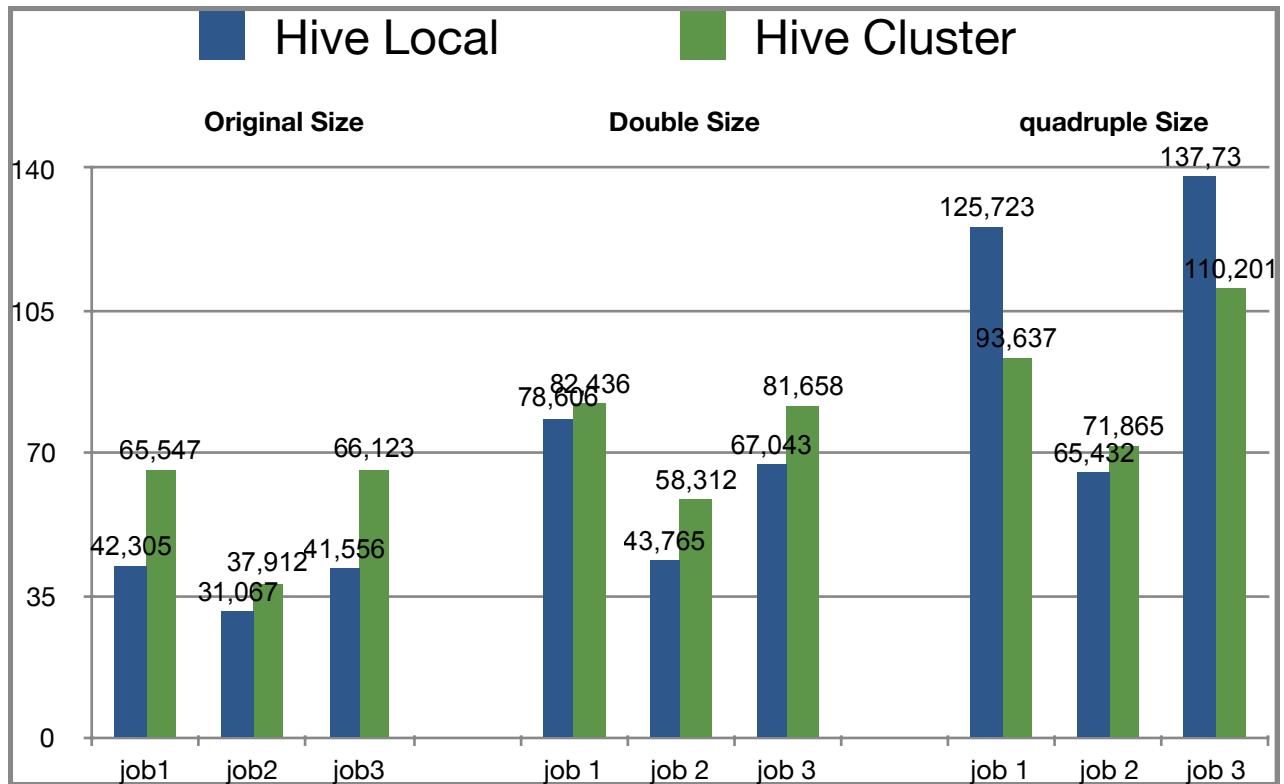
I seguenti grafici sono stati suddivisi per le diverse tecnologie utilizzate Map-Reduce, Hive e Spark.

Inoltre i diversi tempi rappresentano le medie calcolate per ogni job e suddivise per dimensione dell'input (i file con dimensioni maggiori sono stati generati copiando più volte il file dato).

## MAP REDUCE



## HIVE



## SPARK

