

Développer des programmes map reduce avec Python

Exercice 1. Créez un nouveau fichier et créer votre premier programme permettant de compter les fréquences de chaque rating (note: nombre d'étoiles) utilisés pour noter les films (movies) dans le fichier u.data

Extrait du fichier u.data (l'entête est ajoutée pour clarification)

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488

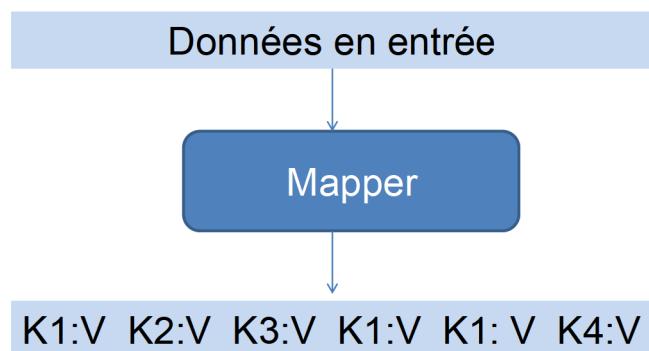
1

2

Comment MapReduce fonctionne?

Mapping

Convertir les lignes de données sous forme de paires
Clé/valeur



Rôle du Mapper

196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488

```
class MRRatingCounter(MRJob):  
    def mapper(self, key, line): (userID, movieID, rating, timestamp)=line.split('\t')  
    yield rating, 1
```

3

3:1 3:1 1:1 2:1 1:1 4:1 2:1

4

Grouper et trier

3:1 3:1 1:1 2:1 1:1 4:1 2:1



1:1,1 2:1,1 3:1,1 4:1

Rôle du reducer

1:1,1 2:1,1 3:1,1 4:1



```
def reducer(self, rating, occurrences):
    yield rating, sum(occurrences)
```

1:2 2:2 3:2 4:1

5

```
from mrjob.job import MRJob
class MRRatingCounter(MRJob):
    def mapper(self, key, line):
        (userID, movieID, rating, timestamp) = line.split("\t")
        yield rating, 1

    def reducer(self, rating, occurrences):
        yield rating, sum(occurrences)

if __name__ == '__main__':
    MRRatingCounter.run()
```

Mettre tous ensembles

196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488

```
class MRRatingCounter(MRJob):
    def mapper(self, key, line):
        (userID, movieID, rating, timestamp)=line.split("\t")
        yield rating, 1
```

3:1 3:1 1:1 2:1 1:1 4:1 2:1



1:1,1 2:1,1 3:1,1 4:1



```
def reducer(self, rating, occurrences):
    yield rating, sum(occurrences)
```

1:2 2:2 3:2 4:1

6

Enregistrer le programme dans le dossier "d:\BigData" que vous venez de créer en donnant au programme le nom "RatingCounter"

Pour exécuter le programme, taper dans l'éditeur inférieur de spyder:

!python RatingCounter.py ml-100k\l.data

Note: par défaut, l'environnement d'exécution pour python est c:\users\computer_name

Il faut modifier l'environnement de travail vers **d:\bigdata** en tapant dans l'éditeur supérieur de Spyder :

cd d:\bigdata

7

8

Resultats

Python	18000101.csv
"1"	6110
"2"	11370
"3"	27145
"4"	34174
"5"	21201

9

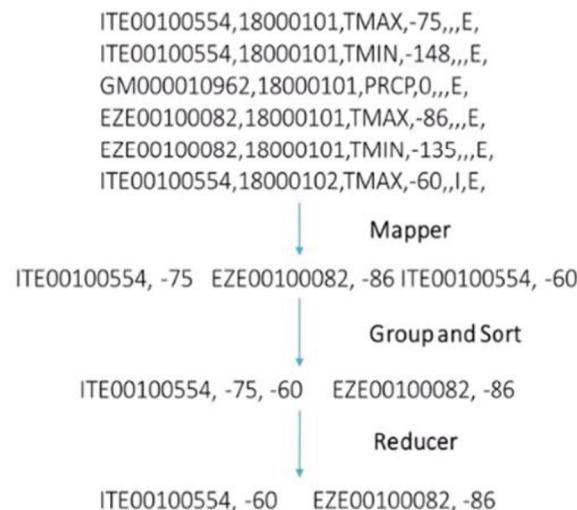
Exercice 2. Ecrire un programme map-reduce qui cherche la température maximale pour chaque région dans un ensemble de données réel de 1825 enregistrements dans un fichier apple 1800.csv.

Ouvrez le fichier 1800.csv et examiner son contenu

```
ITE00100554,18000101,TMAX,-75,,E,  
ITE00100554,18000101,TMIN,-148,,E,  
GM000010962,18000101,PRCP0,,E,  
EZE00100082,18000101,TMAX,-86,,E,  
EZE00100082,18000101,TMIN,-135,,E,  
ITE00100554,18000102,TMAX,-60,,I,E,
```

10

Le rendre un problème map-reduce



```
class MRMaxTemperature(MRJob):  
  
    def mapper(self, _, line):  
        (location, date, type, data, x, y, z, w) = line.split(',')  
        if (type == 'TMAX'):  
            yield location, data  
  
    def reducer(self, location, temps):  
        yield location, max(temps)  
  
if __name__ == '__main__':  
    MRMaxTemperature.run()
```

11

12

Exercice 3. Modifier le programme map-reduce de l'exercice 2 et trouver le maximum de température en fahrenheit pour chaque région.

Nous avons besoin d'une fonction `MakeFahrenheit` qui convertit les températures du celsius à fahrenheit. La conversion est effectuée comme suit:

$$\text{fahrenheit} = \text{Celsius} * 1.8 + 32$$

La température existante dans le fichier est sous forme `celcius*10`

13

```
class MRMaxTemperature(MRJob):

    def MakeFahrenheit(self, tenthsOfCelsius):
        celsius = float(tenthsOfCelsius) / 10.0
        fahrenheit = celsius * 1.8 + 32.0
        return fahrenheit

    def mapper(self, _, line):
        (location, date, type, data, x, y, z, w) = line.split(',')
        if (type == 'TMAX'):
            temperature = self.MakeFahrenheit(data)
            yield location, temperature

    def reducer(self, location, temps):
        yield location, max(temps)

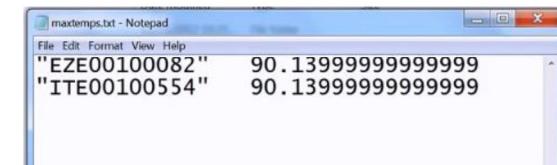
if __name__ == '__main__':
    MRMaxTemperature.run()
```

15

```
def MakeFahrenheit(self, tenthsOfCelsius):
    Celsius = float(tenthsOfCelsius) / 10.0
    fahrenheit = celsius * 1.8 + 32.0
    return fahrenheit
```

14

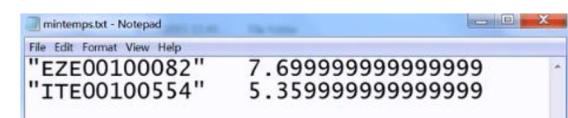
Résultats



"EZE00100082"	90.13999999999999
"ITE00100554"	90.13999999999999

Exercice 4. écrire un programme map-reduce qui cherche la température minimale en fahrenheit par region en utilisant le même fichier de l'exercice 3.

Résultats



"EZE00100082"	7.699999999999999
"ITE00100554"	5.359999999999999

16

Le rendre un problème map-reduce

Exercise 5. Ecrire un programme map-reduce qui calcule la moyenne des amis par âge.

Extrait du fichier friends.csv Quels sont les attributs?

```
0,Will,33,385
1,Jean-Luc,26,2
2,Hugh,55,221
3,Deanna,40,465
4,Quark,68,21
5,Weyoun,59,318
6,Gowron,37,220
7,Will,54,307
8,Jadzia,38,380
9,Hugh,27,181
10,Odo,53,191
```

User ID, Name, Age, Number
of Friends

User ID, Name, Number of Friends

Mapper

Age, Number of Friends

Group and Sort

Age, # Friends, # Friends Age, # Friends, #Friends, #Friends ...

Reducer

Age, Average # of Friends Age, Average # of Friends ...

17

18

```
from mrjob.job import MRJob
class MRFriendsByAge(MRJob):
```

```
    def mapper(self, _, line):
        (ID, name, age, numFriends) = line.split(',')
        yield age, int(numFriends)

    def reducer(self, age, numFriends):
        total = 0
        numElements = 0
        for x in numFriends:
            total += x
            numElements += 1
        yield age, int(total / numElements)
if __name__ == '__main__':
    MRFriendsByAge.run()
```

Le reducer peut être fait avec une autre méthode comme suit:

```
def reducer(self, age, numFriends):
    n=list(numFriends)
    s=sum(n)
    l=len(n)
    yield age, s/l
```

19

20

Exécuter le programme en tapant dans l'éditeur en bas:

```
!python FreindsByAge.py freinds.csv > friendsbyage.txt
```

Résultats du programme sera sauvegarder dans le fichier `friendsbyage.txt`

	File	Edit	Format	View	Help
"18"	343.375				
"19"	213.27272727272728				
"20"	165.0				
"21"	350.875				
"22"	206.42857142857142				
"23"	246.3				
"24"	233.8				
"25"	197.45454545454547				
"26"	242.05882352941177				
"27"	228.125				
"28"	209.1				
"29"	215.91666666666666				
"30"	235.8181818181818				
"31"	267.25				
"32"	207.9090909090909				
"33"	325.3333333333333				
"34"	245.5				
"35"	211.625				
"36"	246.6				
"37"	249.33333333333334				

21

22

Exercice 6. écrire un programme map-reduce (WordFrequency.py) qui calcule les fréquences des mots d'un livre (`book.txt`).

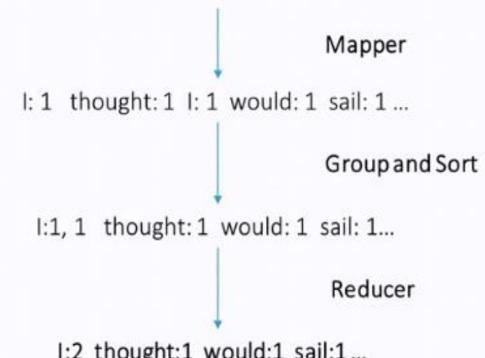
Input Data

Text from a book:

"I thought I would sail about a little and see the watery part of the world."

Making it a MapReduce Problem

I thought I would sail about a little and see the watery part of the world.



23

24

```

from mrjob.job import MRJob
class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        words = line.split(" ")
        for word in words:
            yield word.lower(), 1
    # On ne peut pas écrire: yield words.lower(), 1 sans
    # utiliser la boucle car words est une liste de valeurs
    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()

```

25

Améliorer le programme en ajoutant une expression régulière (RE) qui tient en considération que le mot est constitué uniquement de caractères mots (lettres, chiffres et _) et élimine les signes de ponctuation, etc.:

`WORD_REGEX=re.compile(r"\w'+")`

script définissant une RE

Doit être avant la définition de classe

`\w`: caractère mot: pour les caractères alphanumériques et _ équivalent à l'ensemble `[a-zA-Z0-9_]`

`\w'`: considérer les caractères mots en ignorant les ponctuation, les espaces, etc.

`+`: un ou plusieurs répétitions du RE

"by."	1	
"c-suite."	1	
"c#"	1	
"c++"	1	
"cable"	1	
"cache"	2	
"caching"	1	
"caffeine-fueled"	1	
"calculating"	1	
"calculations"	1	
"calendar"	1	
"california"	1	
"california,"	1	
"call"	8	
"call."	1	
"called"	9	
"calling"	1	
"calls"	1	
"calls,"	3	
"came"	11	
"campaign"	17	
"campaign,"	2	
"campaign."	1	
"campaigns"	12	
"campaigns,"	3	
"campaigns,\\"	1	

"campaigns,"	3	
"campaigns,\\"	1	
"campaigns."	5	
"can't"	22	
"can"	340	
"can,"	3	
"can."	6	
"canada,"	1	
"cannot"	9	
"capabilities"	1	
"capital"	4	
"capital,"	1	
"capitalists"	2	
"capture"	2	
"capture."	1	
"car"	2	
"car),"	1	
"car,)"	1	
"car?"	2	
"card"	1	
"card,"	1	
"cardboard"	1	
"cards"	1	
"cards."	1	
"care"	22	
"care."	1	

Remarquons que le mot « campain » a une fréquence 17, suivi du mot « campain, » de fréquence 2 et « campain. » de fréquence 1. Il s'agit du même mot mais il y'a les signes de ponctuations qui rendent les mots différents.

26

Le package des expressions régulières doit être importé après la première ligne du programme.

`import re`

Dans la définition du mapper:

`Words=WORD_REGEX.findall(line)`

Trouver les mots dans la ligne répondant à l'expression régulière WORD_REGEX.

Exécuter le programme :

`!python WordFrequency.py book.txt > wordcount.txt`

27

28

```

from mrjob.job import MRJob
import re

WORD_REGEX = re.compile(r"\w+")

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        words = WORD_REGEX.findall(line)
        for word in words:
            yield word.lower(), 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()

```

Results:

" "	28
"0"	1
"000"	24
"05"	1
"07"	1
"1"	13
"10"	23
"100"	10
"1000"	1
"101"	2
"104"	1
"1099"	1
"11"	3
"1124"	1
"12"	1
"125"	1

wordsbetter.txt - Notepad	
File	Edit
"82"	1
"85"	1
"9"	11
"90"	3
"93"	1
"99"	2
"a"	1191
"aaron"	3
"abandon"	1
"abandoned"	1
"abbreviation"	1
"abilities"	3
"ability"	15
"able"	22
"about"	202
"above"	5
"absolutely"	1
"absorbed"	1
"abstract"	1
"accelerator"	1
"accept"	9
"accepted"	3
"accepting"	1
"access"	4
"accident"	3
"accompanied"	3

29

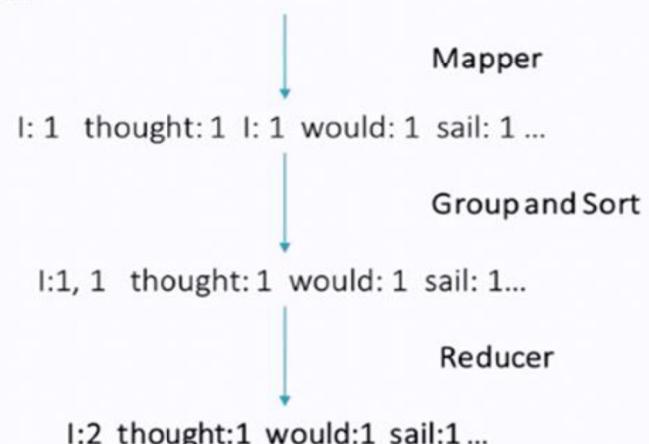
30

Le rendre un problème map-reduce

Utilisation du of Steps dans map-reduce

Exercise 7. Le même programme changera de résultat qui sera trié par la fréquences de mots.

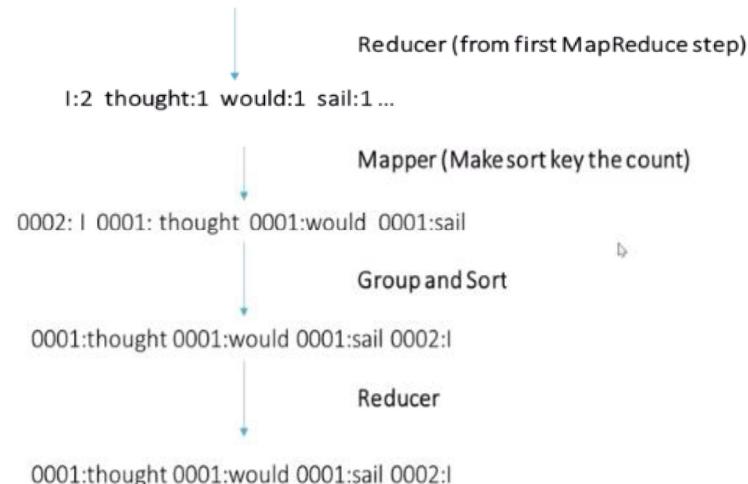
I thought I would sail about a little and see the watery part of the world.



31

32

Trier les résultats finaux: enchaîner les étapes (Steps)



33

La seconde étape map-reduce (2^{ème} MRStep):

La sortie du premier reducer, sera l'entrée du deuxième mapper dans lequel la clé sera word (le mot) et count count est la fréquence du mot. La sortie de ce mapper sera (count, word):

```
yield '%04d'% int(count), word
```

↑
Cast count into int

Présenter count avec 4 digits, ajouter 0 à gauche si nécessaire

35

- Nous avons un « step job », il faut ajouter l'instruction suivante après l'importation de MRJob:

```
from mrjob.Step import MRStep
```

- Nous avons une nouvelle fonction applée steps
- Au lieu d'un mapper et un reducer, nous avons deux mappers et deux reducers.

La première étape map-reduce (1^{ère} MRStep):

- Fonction mapper-get-words pour le mapper (le même code du précédent mapper)
- Fonction reducer-count-words pour le reducer le même code du précédent reducer)

34

```
from mrjob.job import MRJob  
from mrjob.step import MRStep  
import re
```

```
WORD_REGEXP = re.compile(r"[\\w']")
```

```
class MRWordFrequencyCount(MRJob):
```

```
def steps(self):  
    return [  
        MRStep(mapper=self.mapper_get_words,  
               reducer=self.reducer_count_words),  
        MRStep(mapper=self.mapper_make_counts_key,  
               reducer = self.reducer_output_words)  
    ]
```

36

```

def mapper_get_words(self, _, line):
    words = WORD_REGEX.findall(line)
    for word in words:
        yield word.lower(), 1

def reducer_count_words(self, word, values):
    yield word, sum(values)

def mapper_make_counts_key(self, word, count):
    yield '%04d' % int(count), word

def reducer_output_words(self, count, words):
    for word in words:
        yield count, word

if __name__ == '__main__':
    MRWordFrequencyCount.run()

```

37

For the second reducer, the key will be the count, then the results will be sorted by the count

Run the program, put the results in a file named:
wordssorted.txt

Results:

Count	Word
"0255"	"time"
"0278"	"or"
"0280"	"this"
"0315"	"with"
"0321"	"have"
"0334"	"i"
"0343"	"as"
"0354"	"can"
"0369"	"be"
"0382"	"business"
"0411"	"if"
"0424"	"are"
"0428"	"on"
"0496"	"it"
"0537"	"for"

38

Exercise 8. Ecrire un programme map-reduce qui calcule le total de “order amounts” par un client (customer)

Extrait du fichier customer-orders.csv

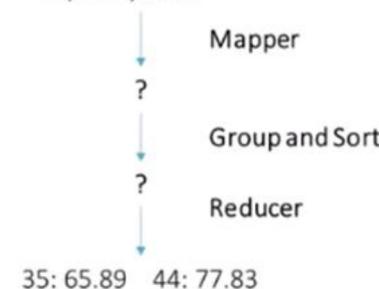
44,8602,37.19
35,5368,65.89
44,3391,40.64
47,6694,14.98
29,680,13.08
91,8900,24.59
70,3959,68.68

Quels sont les attributs?

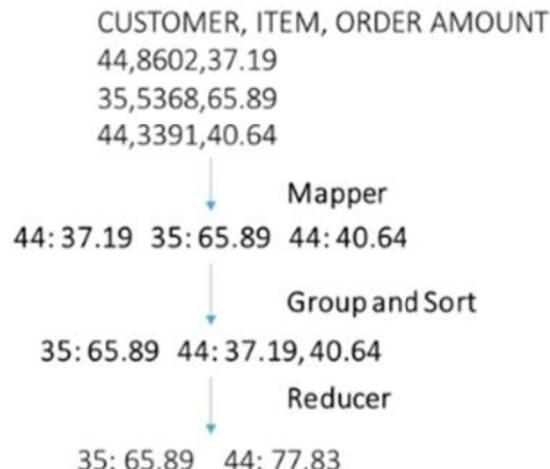
CUSTOMER, ITEM, ORDER AMOUNT

39

CUSTOMER, ITEM, ORDER AMOUNT
44,8602,37.19
35,5368,65.89
44,3391,40.64



40



```

from mrjob.job import MRJob

class SpendByCustomer(MRJob):

    def mapper(self, _, line):
        (customerID, itemID, orderAmount) = line.split(',')
        yield customerID, float(orderAmount)

    def reducer(self, customerID, orders):
        yield customerID, sum(orders)

if __name__ == '__main__':
    SpendByCustomer.run()

```

41

42

Results:

```

ordertotals.txt - Notepad
File Edit Format View Help
"0"      5524.9500000000000002
"1"      4958.6000000000001
"10"     4819.7
"11"     5152.29
"12"     4664.589999999999
"13"     4367.619999999999
"14"     4735.0300000000001
"15"     5413.5100000000001
"16"     4979.0600000000001
"17"     5032.6800000000001
"18"     4921.27
"19"     5059.429999999998
"2"      5994.5900000000001
"20"     4836.86
"21"     4707.409999999999
"22"     5019.45

```

Pour afficher l'ordre total dépensé par un client (customer) comme un nombre réel avec deux nombres décimaux après le point décimal et quatre digits avant le point décimal: `'%04.02f'%float(sum(orders))`

Exercise 10. Trier le résultat de l'exrcice précédent suivant les totaux dépensés par les clients.

43

44

```
from mrjob.job import MRJob  
from mrjob.step import MRStep
```

```
class SpendByCustomerSorted(MRJob):  
    def steps(self):  
        return [  
            MRStep(mapper=self.mapper_get_orders,  
                   reducer=self.reducer_totals_by_customer),  
            MRStep(mapper=self.mapper_make_amounts_key,  
                   reducer=self.reducer_output_results)  
        ]  
  
    def mapper_get_orders(self, _, line):  
        (customerID, itemID, orderAmount) = line.split(',')  
        yield customerID, float(orderAmount)
```

45

```
def reducer_totals_by_customer(self, customerID, orders):  
    yield customerID, sum(orders)
```

```
def mapper_make_amounts_key(self, customerID, orderTotal):  
    yield '%04.02f' %float(orderTotal), customerID
```

```
def reducer_output_results(self, orderTotal, customerIDs):  
    for customerID in customerIDs:  
        yield orderTotal, customerID
```

```
if __name__ == '__main__':  
    SpendByCustomerSorted.run()
```

46

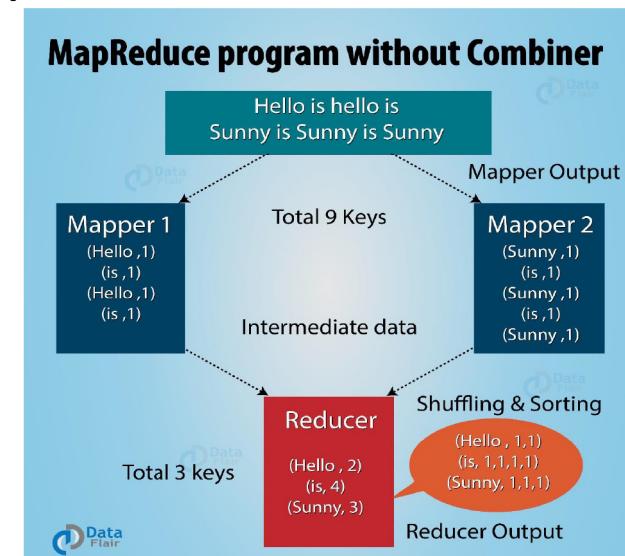
Utiliser un combineur en map-reduce

Le combiner est utilisé entre le Mapper et le Reducer pour réduire le volume de données transféré entre eux. Toujours la sortie de la tâche map est large et les données transférées à la tâche reduce sont élevées.

- Le combiner opère sur chaque clé sortie du mapper. Il doit avoir la même sortie clé, valeur que le reducer.
- Le combiner produit un résumé d'information à partir d'un large ensemble de données sorties du mapper.

47

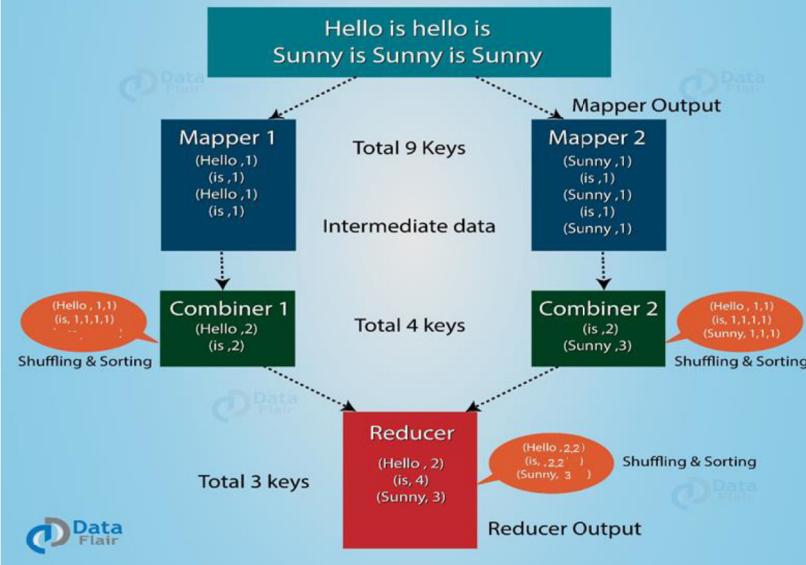
Exemple:



<https://data-flair.training/blogs/hadoop-combiner-tutorial/>

48

MapReduce program with Combiner

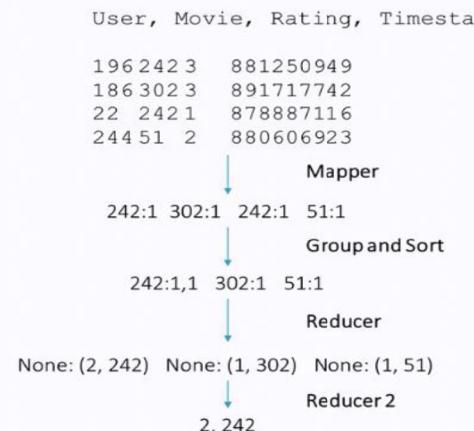


<https://data-flair.training/blogs/hadoop-combiner-tutorial/>

49

Exercise 12. écrire un programme map-reduce qui trouve le film le plus noté à partir du fichier u.data

Most-Rated Movie



Results:

In [2]: !python MostPopularMovie.py ml-100k/u.data
583 "50"

51

Exercise 11. réécrire le programme de fréquence de mot avec les combineurs

```
from mrjob.job import MRJob
class MRWordFrequencyCount(MRJob):
```

```
def mapper(self, _, line):
    words = line.split()
    for word in words:
        yield word.lower(), 1
```

```
def combiner(self, key, values):
    yield key, sum(values)
```

```
def reducer(self, key, values):
    yield key, sum(values)
```

```
if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

50

```
1 # -*- coding: utf-8 -*-
2 from mrjob.job import MRJob
3 from mrjob.step import MRStep
4
5 class MostPopularMovie(MRJob):
6     def steps(self):
7         return [
8             MRStep(mapper=self.mapper_get_ratings,
9                    reducer=self.reducer_count_ratings),
10            MRStep(
11                reducer = self.reducer_find_max)
12        ]
13
14     def mapper_get_ratings(self, _, line):
15         (userID, movieID, rating, timestamp) = line.split('\t')
16         yield movieID, 1
17
18     def reducer_count_ratings(self, key, values):
19         yield None, (sum(values), key)
20
21     def reducer_find_max(self, key, values):
22         yield max(values)#le tri sera effectué selon la première valeur du tuple
23
24 if __name__ == '__main__':
25     MostPopularMovie.run()
```