MapReduce

Un modèle de programmation pour les calculs distribués

M. Ben & R. Tavenard

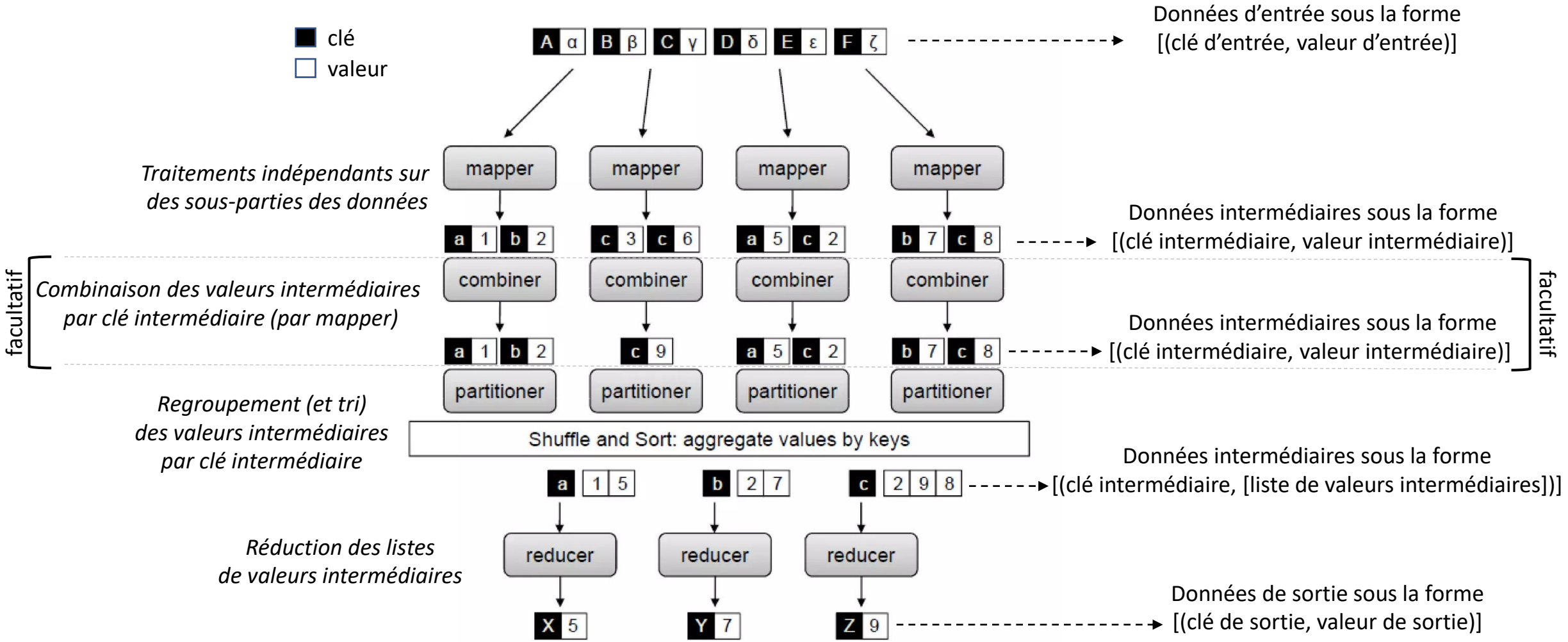
Le modèle de programmation MapReduce

- Modèle de programmation adapté au traitement de données très volumineuses (big data)
  - Implémenté dans le framework Hadoop

- Principe :
  - Étape « Map » : effectuer des (sous-)traitements indépendants
    - Possibilité de les paralléliser et/ou de les distribuer
  - Étape « Reduce » : rassembler et agréger les résultats des traitements indépendants pour obtenir le résultat final

- Flux de traitements des données basé sur des listes de couples `(clé,valeur)`

Flux de traitements MapReduce



Source de l'image : https://www.slideshare.net/subhaskghosh/04-combiner-and-partitioner

Flux de traitements MapReduce

- Remarques
  - Les clés d'entrée peuvent être simplement des indices dans l'ensemble des données d'entrée et ne sont pas forcément utilisées par la suite.
  - Les `combiner` se font `mapper` par `mapper`
    - Etape de pré-réduction au niveau de chaque `mapper` (ils peuvent être intégrés au traitement du `mapper`)
    - Permettent de diminuer la quantité de données à envoyer au `partitioner`
    - Facultatifs : s'ils sont omis, toutes les combinaisons et réductions se font en amont par le `partitionner` et les `reducer`
  - Si plusieurs réductions indépendantes sont attendues (pour obtenir plusieurs valeurs de sortie) possibilité de paralléliser et/ou distribuer les `reducer` → étape de type « Map » sur les `reducer`

Flux de traitements MapReduce

- Exemple de flux pour le problème du « wordcount »

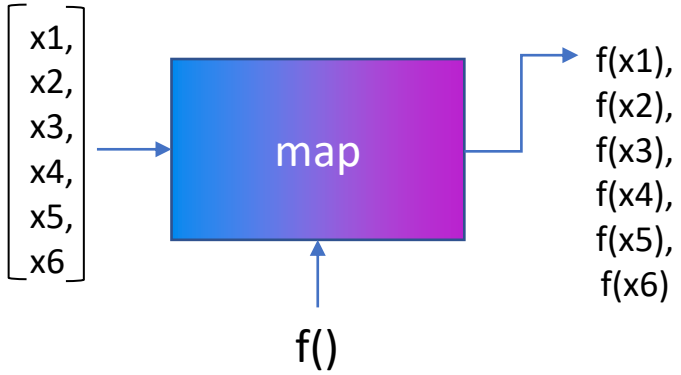
  → On dispose d'un corpus de textes réparti en plusieurs fichiers
  → On veut compter le nombre d'occurrences de chaque mot dans le corpus

- (clé, valeur) d'entrée = (indice d'un fichier, nom du fichier)
  - `mapper(+combiner)` = *fonction de comptage des mots dans **un** fichier*
- (clé, valeur) intermédiaire = (mot, comptage du mot dans **un** fichier)
  - `partitioner` = *fonction qui rassemble les comptages (dans chaque fichier) par mot*
- (clé, [liste de valeurs]) intermédiaire = (mot, [**liste** des comptages du mot dans chaque fichier])
  - `reducer` = *fonction qui additionne les comptages d'**un** mot*
- (clé, valeur) de sortie = (mot, comptage du mot dans le corpus)

Remarque : ici il y a un `reducer` *par mot. Ces* `reducer` *peuvent être parallélisés par un processus de type « Map »*

Les fonctions `map()` et `reduce()` de Python

- `map()`
  - Applique une fonction à chaque élément d'un (ou plusieurs) itérable(s)
  - Renvoie un **itérateur** sur les résultats




```
def incr(x):
    return x+1

l = [0, 1, 2, 3, 4]

list(map(incr, l))
```

✓ 0.0s

```
[1, 2, 3, 4, 5]
```

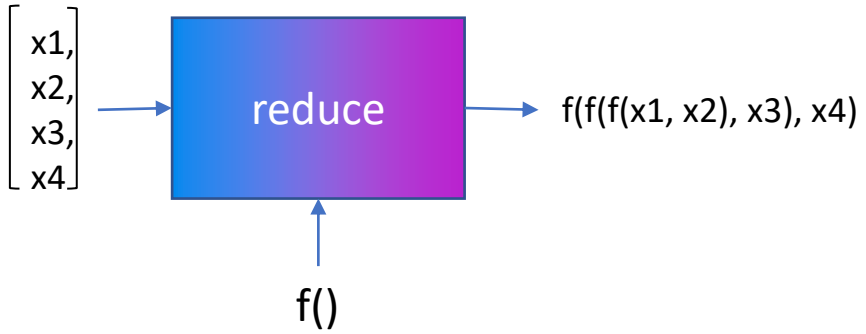
```
def comp(x1, x2):
    return x1 > x2

l1 = [2, 1, 9, 2, 3]
l2 = [5, 1, 8, 0, 6]

list(map(comp, l1, l2))
```

✓ 0.0s

```
[False, False, True, True, False]
```

Les fonctions `map()` et `reduce()` de Python

- `reduce()` du module `functools`
  - Applique une fonction de 2 arguments de façon récursive à l'ensemble des éléments d'un itérable.
  - Exemple :
    - Fonction = `f()`
    - Éléments de l'itérable = `x1,x2,x3,x4`
    - Résultat = `f(f(f(x1, x2), x3), x4)`

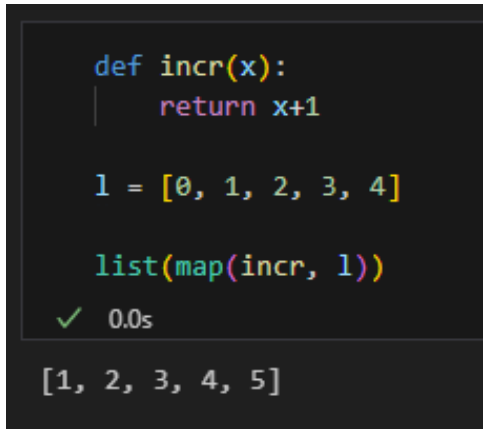
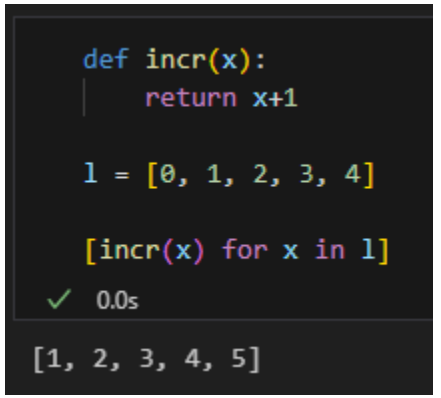
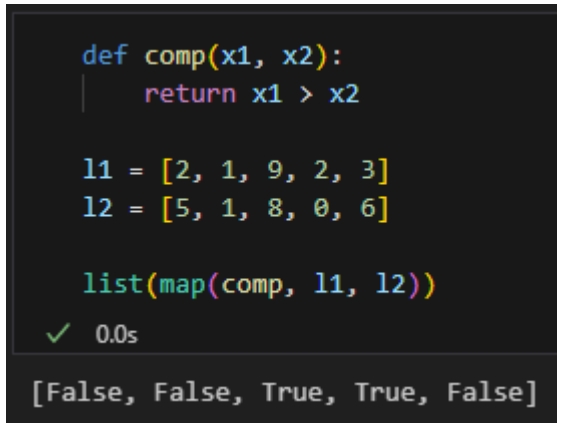
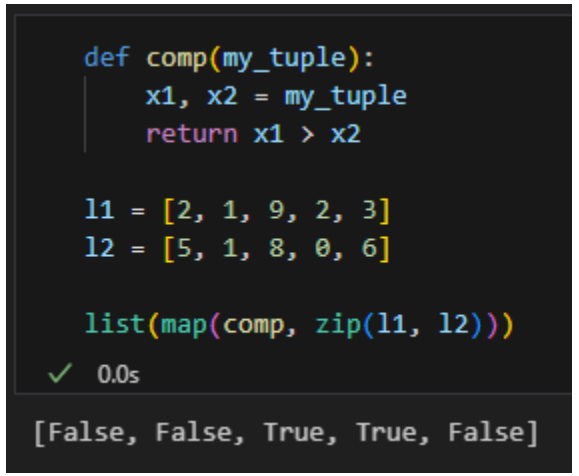



```
from functools import reduce

def mon_max(a, b):
    return a if a>=b else b

l = [12, 2, 0 , 15 , 3, 10]

reduce(mon_max, l)
```

✓ 0.0s

```
15
```

Les fonctions `map()` et `reduce()` de Python

- Remarques
  - 
```
def incr(x):
    return x+1

l = [0, 1, 2, 3, 4]

list(map(incr, l))
```
✓ 0.0s
```
[1, 2, 3, 4, 5]
```
est équivalent à

```
def incr(x):
    return x+1

l = [0, 1, 2, 3, 4]

[incr(x) for x in l]
```
✓ 0.0s
```
[1, 2, 3, 4, 5]
```
  - 
```
def comp(x1, x2):
    return x1 > x2

l1 = [2, 1, 9, 2, 3]
l2 = [5, 1, 8, 0, 6]

list(map(comp, l1, l2))
```
✓ 0.0s
```
[False, False, True, True, False]
```
est équivalent à

```
def comp(my_tuple):
    x1, x2 = my_tuple
    return x1 > x2

l1 = [2, 1, 9, 2, 3]
l2 = [5, 1, 8, 0, 6]

list(map(comp, zip(l1, l2)))
```
✓ 0.0s
```
[False, False, True, True, False]
```