Bonjour

Rappels (HDFS)

Jusqu'ici

- ➔ On a présenté HDFS, le système de fichiers distribués de Hadoop.
- ➔ On a vu comment fonctionne l'architecture, comment sont stockées les données.
- ➔ Comment interagir entre notre machine locale et HDFS

Maintenant, on sait comment stocker. Mais comment traiter ces données ?

MAP-REDUCE (1)

Modèles de programmation large échelle (Map-Reduce)

Comment traiter les données distribuées

- ➔ Google a inventé MapReduce afin de permettre à leur moteur de recherche de passer à l'échelle
- ➔ Idée de MapReduce :
  - ◆ On écrit des fonctions qui vont s'exécuter localement sur chacune des machines
  - ◆ On récupère les résultats intermédiaires, on les agrège pour avoir une valeur finale

- ➔ Cela peut paraître très restreignant car il faut exprimer tous ces algorithmes à l'aide de deux fonctions map et reduce
- ➔ Mais en pratique, on arrive à presque tout faire avec et ça passe à l'échelle

Inspiré largement du paradigme diviser pour régner

- Diviser : découper le problème initial en sous problèmes
- Régner : résoudre les sous-problèmes indépendamment
- Combiner : construire la solution du problème initial en combinant les solutions des différents sous problèmes

➔ Mapreduce c'est **Diviser pour distribuer pour régner**

➔ Les données sont distribuées sur plusieurs machines,

Inspiration fonctionnelle

- **map** : consiste à appliquer une même fonction à tous les éléments de la liste

  map(f)[x0, ..., xn] = [f(x0, ...,f(xn)]

  map(*4)[2, 3, 6] = [8, 12, 24]

- **reduce** applique une fonction récursivement à une liste et retourne un seul résultat

  reduce(f)[x0, ..., xn] = f(x0, f(x1, f(x2, ...)))

  reduce(+)[2, 3, 6] = (2 + (3 + 6)) = 11

Map-Reduce

Un modèle de programmation, pour traiter les données distribuées dans un cluster.

Souvent, le but est de récupérer une information synthétique à partir d'un jeu de données

➔ Calculer le montant total des ventes d'un article
➔ Trouver l'article le plus cher
➔ Calculer le prix moyen dépensé par chaque client

Chacun de ses exemples peut s'écrire sous la forme de la composition de deux fonctions : map et reduce

Les données sont toujours représentées par des paires (clé, valeur)

➔ Les données échangées sont des paires (key, value):

- Une clé : n'importe quel type de données : entier, texte..
- Une valeur : pareil

➔ Un programme map reduce reçoit des paires (clé, valeur) et émet d'autres paires, selon les besoins de l'algorithme

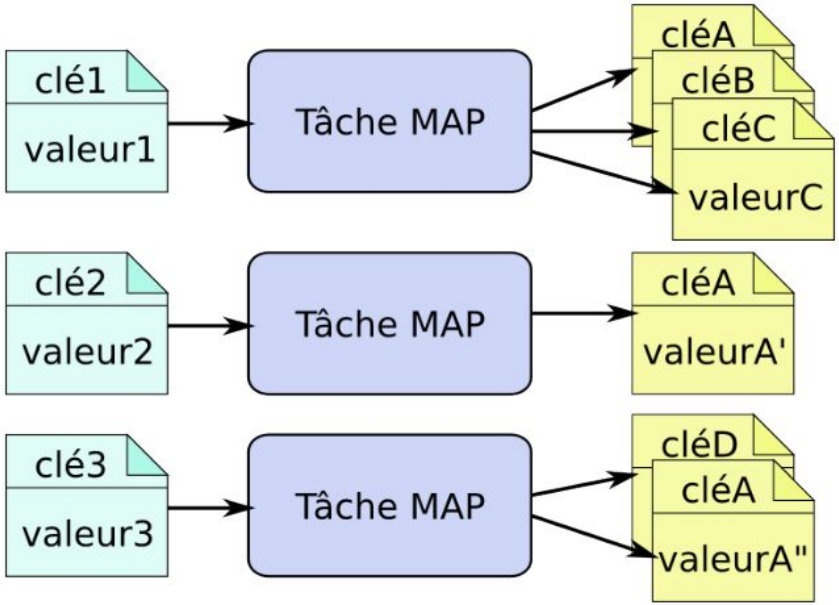
EPITA

Map

➔ La fonction Map reçoit une paire en entrée et peut produire un nombre quelconque de paires en sortie : aucune, une ou plusieurs, à volonté. Les types des entrées et des sorties sont comme on veut.

Schéma de Map

Chaque tâche Map traite une paire et produit 0..n paires

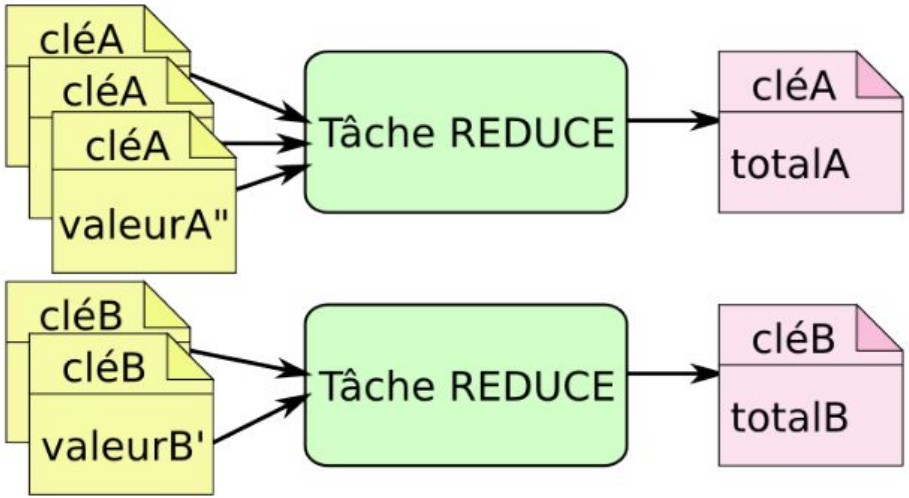


EPITA

Reduce

- Reduce est une fonction reçoit une liste de paires en entrée
- C'est les paires qui sont produites par l'opération Map qui lui correspond
- Reduce produit un nombre quelconque de paires en sortie (souvent 1 seule)
- Le point important c'est que les paires d'entrées de la fonction reduce ont toutes la même clés
- Cette opération est invisible pour l'utilisateur (C'est hadoop (YARN) qui s'occupe de cela, c'est l'opération Shuffle and sort)
- En général, un reducer fait un traitement sur les valeurs, comme une somme, un max, ou une autre fonction d'agrégation qu'on aura définit

Schéma du Reduce



Étapes d'un job MapReduce

1. Prétraitement des données d'entrée
2. **Split** : séparation des données en blocs traitables mis sous forme de (clé, valeur). ex: numéro de ligne, texte de la ligne
3. **Map :** application de la fonction map sur toutes les paires (clé, valeur) formées à partir des données en entrée
4. **Shuffle&Sort** : redistribution des données afin que les paires produites par l'opération Map ayant les mêmes clés soient sur les même machines
5. **Reduce :** agrégation des paires ayant la même clé pour obtenir un résultat

WordCount le helloworld de Map-Reduce

Word Count !

Compter le nombre d'occurrences de chaque mot dans un ensemble de textes.

Données : un ensemble de textes

Le jour se lève sur notre grisaille, sur les trottoirs de nos ruelles et sur nos tours
[...]

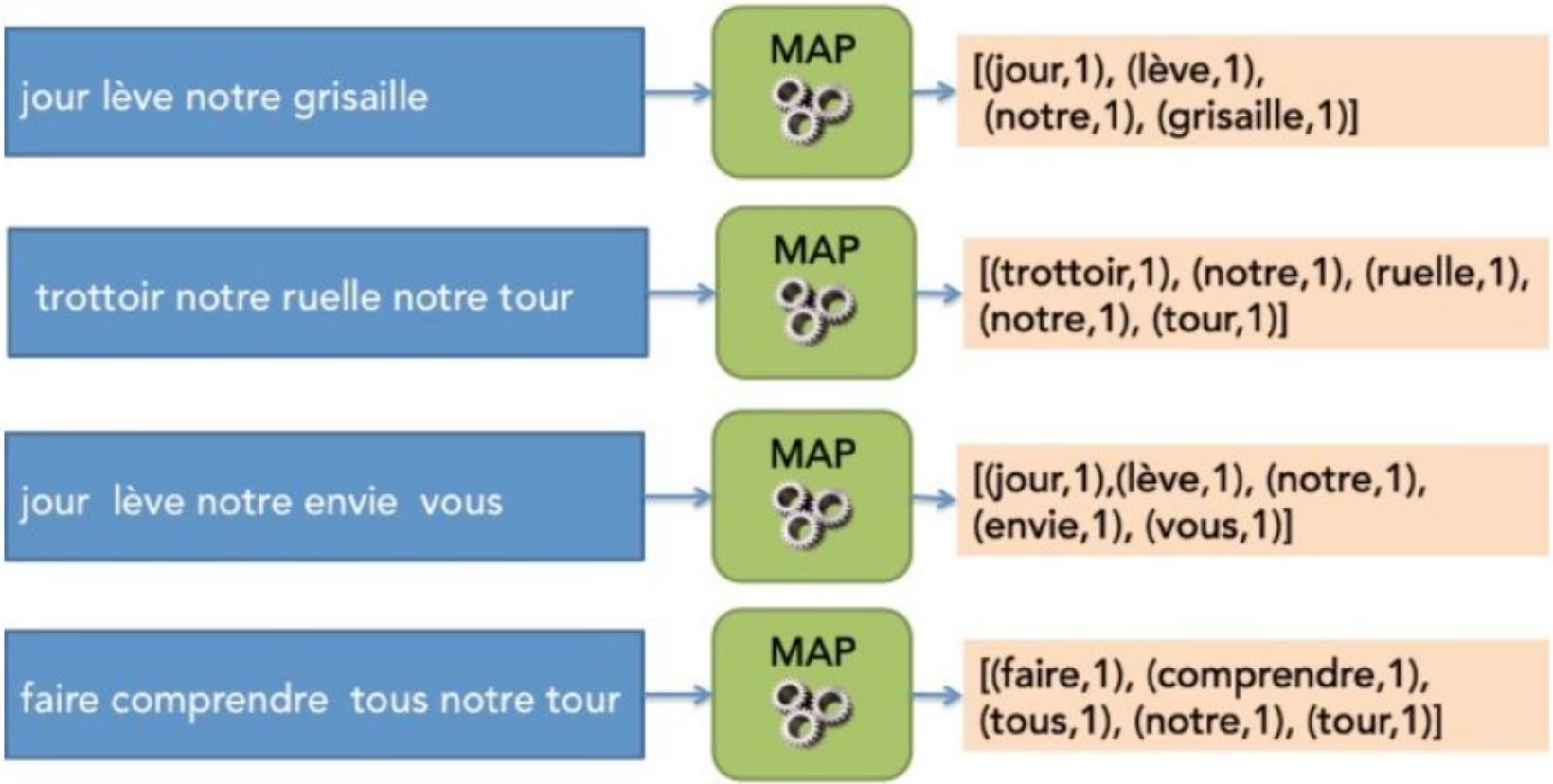
Le jour se lève sur notre envie de vous faire comprendre à tous que c'est à notre tour
[...]

EPITA

Le Mapper

```
def map(key,value):
    intermediate=[]
    for word in value.split():
        intermediate.append((word, 1))
    return intermediate
```

Résultas


jour lève notre grisaille
MAP
[(jour,1), (lève,1), (notre,1), (grisaille,1)]
trottoir notre ruelle notre tour
MAP
[(trottoir,1), (notre,1), (ruelle,1), (notre,1), (tour,1)]
jour lève notre envie vous
MAP
[(jour,1),(lève,1), (notre,1), (envie,1), (vous,1)]
faire comprendre tous notre tour
MAP
[(faire,1), (comprendre,1), (tous,1), (notre,1), (tour,1)]

Shuffle and Sort

(comprendre, [1])

(envie,[1])

(faire,[1])

(grisaille,[1])

(jour,[1,1])

(lève, [1,1])

(notre, [1,1,1,1,1])

(ruelle,[1])

(tour,[1,1])

(tous,[1])

(trottoir,[1])

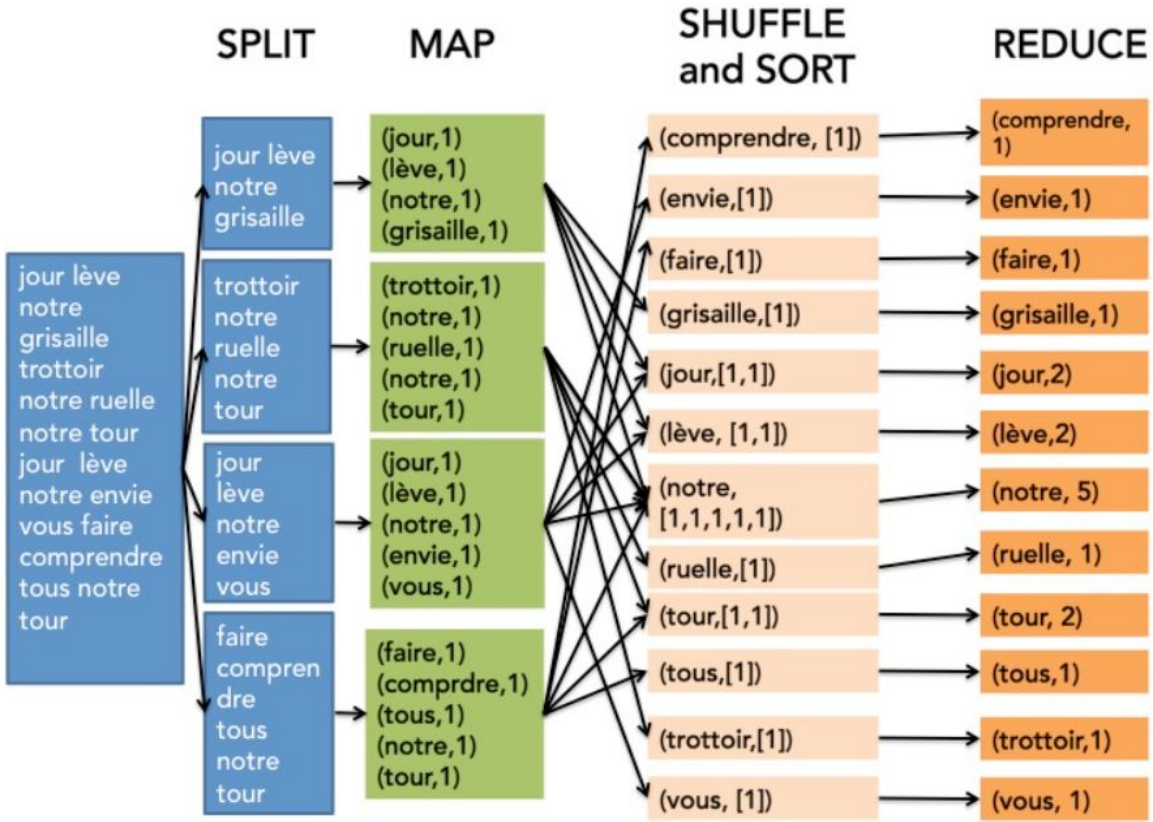
(vous, [1])

EPITA

Le reducer

```
def reduce(key, values):
    result = 0
    for c in values:
        result = result +c
    return (key, result)
```

The Big Picture



EPITA

YARN

Qu’est ce que YARN

YARN (Yet Another Resource Negociator) est un mécanisme dans Hadoop permettant de gérer des jobs sur un cluster

Il permet aux utilisateurs de lancer des jobs MapReduce sur des données présentes dans HDFS

Suivre leur avancement, récupérer des messages (logs)

Peut déplacer un processus d’une machine à une autre en cas de défaillance

YARN est transparent pour l’utilisateur. On lance l'exécution d’un programme MapReduce et YARN s’assure qu’il soit exécuté le plus rapidement possible

EPITA

MRJOB

MRJOB :

- Ecrire des jobs mapreduce en Python
- Librairie Open-source (pip3 install mrjob), maintenu par Yelp
- Bien documenté
- Offre la possibilité d'exécution en mode local ou sur hadoop ou sur Amazon EMR
- https://mrjob.readthedocs.io/en/latest/

Exemple

```python
from mrjob.job import MRJob


class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

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


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

EPITA

Merci !


EPITA
ÉCOLE D'INGENIEURS EN INFORMATIQUE