

Map-Reduce : un pattern de traitement pour le BigData

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

Le Map-Reduce (Google, OSDI 2004)

Un schéma standard

- Un flux de données basé sur la lecture et la production de clé/valeur
- 2 fonctions à programmer : **map** et **reduce**

Un environnement d'exécution

- Automatisation de la parallélisation sur un ensemble d'unités de calcul :
 - distribution des traitements
 - distribution des données
- Équilibrage de charge
- Stockage et transfert de données
- Gestion des éventuelles fautes et de la communication entre machines

⇒ **transparent pour le programmeur**

Flux de données global du Map-Reduce

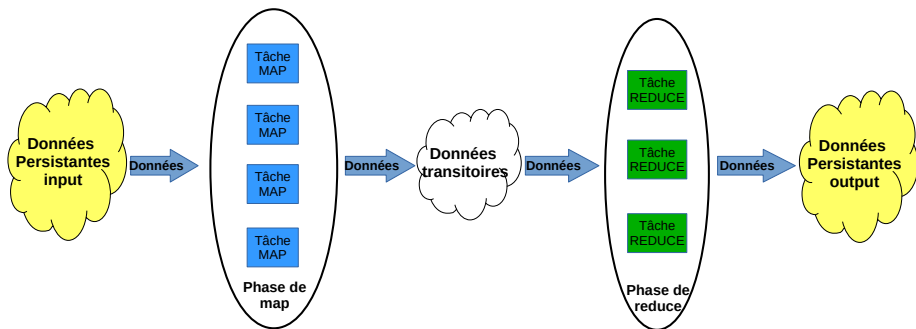


Schéma d'un calcul Map-Reduce

La phase de map

- Lit les données d'entrée sous la forme de $\langle \text{clé}, \text{valeur} \rangle$
- Fait un traitement (exemple : extraire une information précise)
- Produit des données de sortie sous la forme de $\langle \text{clé}, \text{valeur} \rangle$

La phase intermédiaire (Shuffle)

- Transfert, tri et fusionne les données entre les maps et les reduces

La phase de reduce

- Lit les données des maps via le shuffle sous la forme $\langle \text{clé}, \text{valeur} \rangle$
- Fait un traitement (exemple : somme, groupement, filtre, ...)
- Produit les données de sortie du calcul sous la forme de $\langle \text{clé}, \text{valeur} \rangle$

⇒ Le programmeur doit au minimum fournir les fonctions map et reduce pour que son programme fonctionne

Caractéristiques

- Chaque tâche map traite sa partie des données d'entrée appelée **split**
- Chaque élément du split est associé à une clé de type $K1$
- À chaque clé de type $K1$ lue depuis le split, la tâche map correspondante fait un appel à la fonction `map()`.
- La fonction `map()` produit dans le flux d'information une liste de $\langle \text{clé}, \text{valeur} \rangle$ intermédiaire de type $\langle K2, V2 \rangle$
- Nombre de tâches de map = nombre de splits

Map : $(K1, V1) \rightarrow \text{list}(K2, V2)$

Caractéristiques

- Une fois la phase de map terminée, agrégation en liste de toutes les valeurs intermédiaires de type $V2$ associées à une clé de type $K2$.
- À chaque clé de type $K2$ la tâche reduce correspondante fait un appel à la fonction `reduce()`.
- La fonction `reduce()` produit dans le flux d'information une liste de $\langle \text{clé}, \text{valeur} \rangle$ de type $\langle K3, V3 \rangle$
- Chaque paire $\langle K3, V3 \rangle$ émise est enregistrée dans l'ensemble de données de sortie
- Le nombre de tâches de reduce est défini statiquement par l'utilisateur

Reduce : $(K2, \text{list}(V2)) \rightarrow \text{list}(K3, V3)$

Remarque : bien souvent $K2 = K3$

Exemple de calcul : le word-count

- **En entrée** : un ou plusieurs (gros) fichiers textes
- **En sortie** : le nombre d'occurrences de chaque mot du texte

Apple Orange
Banana Peach
Orange Apple
Strawberry
Orange Apple

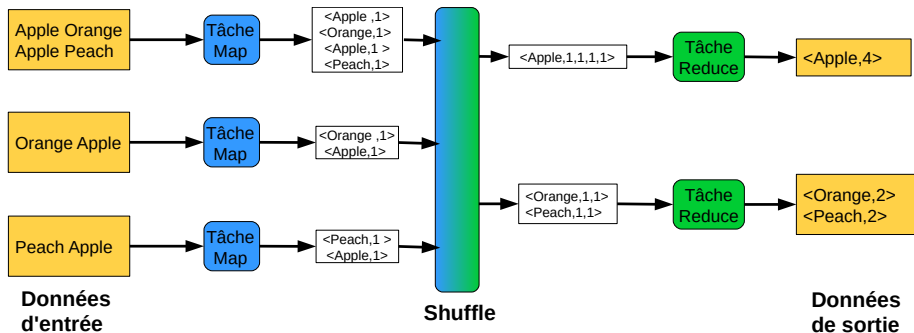
**Données
d'entrée**

word-count 

Apple 3
Banana 1
Peach 1
Orange 3
Strawberry 1

**Données
de sortie**

Exemple de flux de données avec le wordcount



Les fonctions Map et Reduce du wordcount

```
void Map(integer key, string value) {  
    //key : id of the line  
    //value : content of the line  
    for each word w in value {  
        Emit(w, 1);  
    }  
}  
  
void Reduce(string key, list of integer values) {  
    //key : a word  
    //value : a list of counter  
    integer count = 0;  
    for each v in values {  
        count += v;  
    }  
    Emit(key, count);  
}
```

La phase de Shuffle

Transmettre des données de la phase map vers la phase reduce

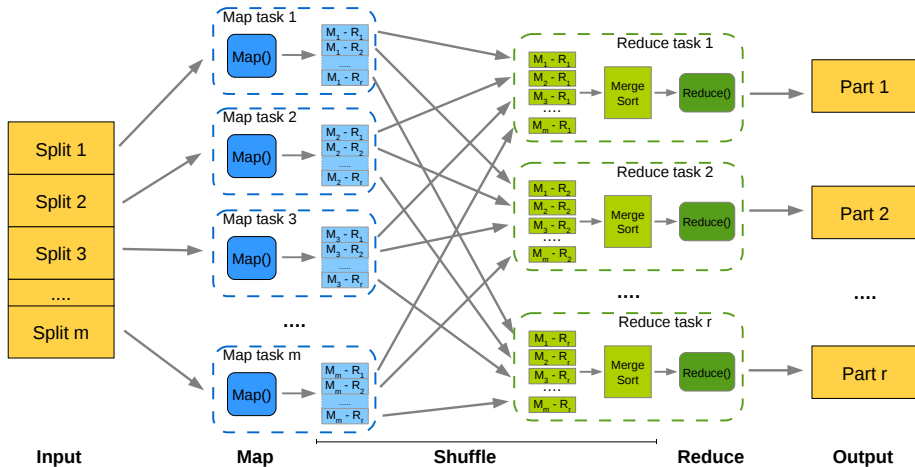
Responsabilité des maps

- stockage local partitionné des couples clé/valeur de sortie de map
- **Partitionnement** : aiguillage déterministe des clés sur $NbReduce$ partitions
⇒ une valeur de clé est associée à un unique reduce

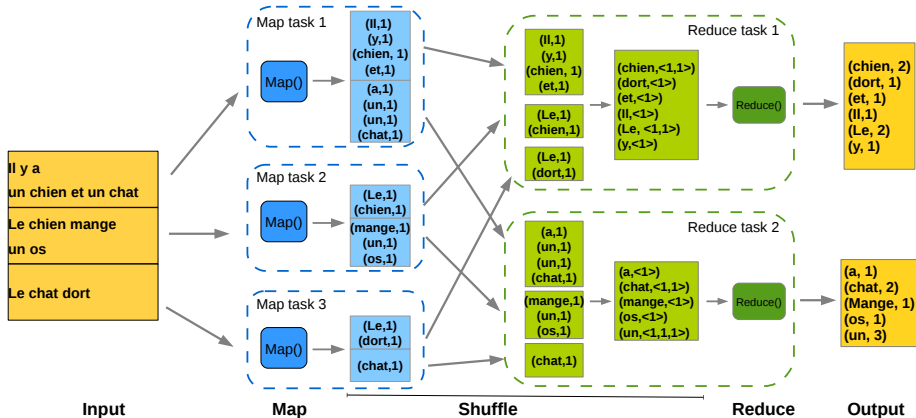
Responsabilité des reduces

- **copy**
 - téléchargement sur chaque map de la partition qui lui est associée
- **merge**
 - agrégation de l'ensemble des partitions téléchargées
 - agrégation des valeurs pour une clé donnée
- **sort**
 - tri des différentes clés définissant l'ordre de lecture par le reduce :
⇒ **un ordre doit être défini pour chaque type de clé**

IMPORTANT



Flux de données détaillé du WordCount



Ce qui **doit** être fourni à l'environnement d'exécution

- une fonction de map
- une fonction de reduce
- des données

Ce qui est configurable

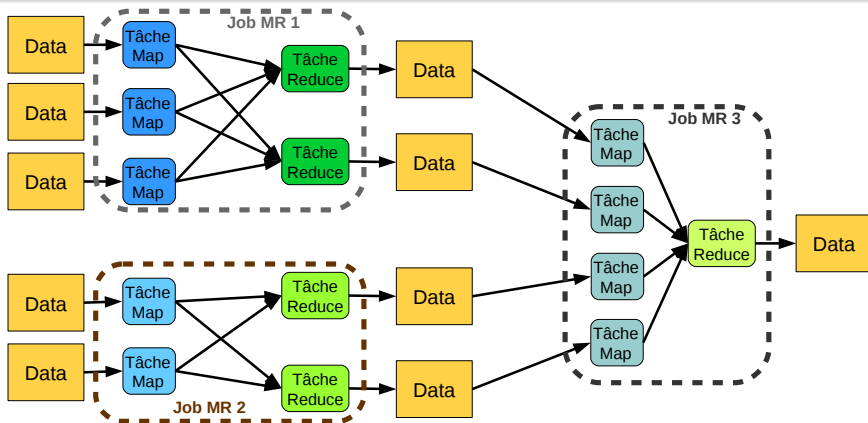
- la politique de partitionnement du shuffle
- la politique de tri du shuffle
- le nombre de tâches de reduce
- la politique de découpage des splits
- les formats d'entrée et de sortie

Enchaînement de flux Map - Reduce

Principe

Les données résultats d'un calcul Map-Reduce peuvent être les données d'entrée d'un autre Map-reduce.

⇒ Possibilité de définir des traitements plus complexes



Traitement SGBD classique vs. traitement Map-Reduce

	SGBD classique	Map-Reduce
Taille	GigaOctets	PetaOctets
Accès	Interactif et batch	batch
Mises à jour	Plusieurs lectures et écritures	Une seule écriture, plusieurs lectures
Données structurées	oui	non
Passage à l'échelle	non linéaire	linéaire

Les différentes implémentations

- Google MapReduce : en C++, propriétaire, API Python et Java
- **Apache Hadoop MapReduce** : en Java, open-source, API Java et adaptateur pour tout langage
- **Apache Spark** : en Scala, open-source, API Scala, Java, Python et R
- Amazon Elastic MapReduce : pour le cloud Amazon
- Microsoft HDInsight : pour le cloud Azur