Gestion des données

Cours 7 – Calcul distribué avec Spark

Olivier Schwander
<olivier.schwander@sorbonne-universite.fr>

2021-2022

Clusters de calcul classiques

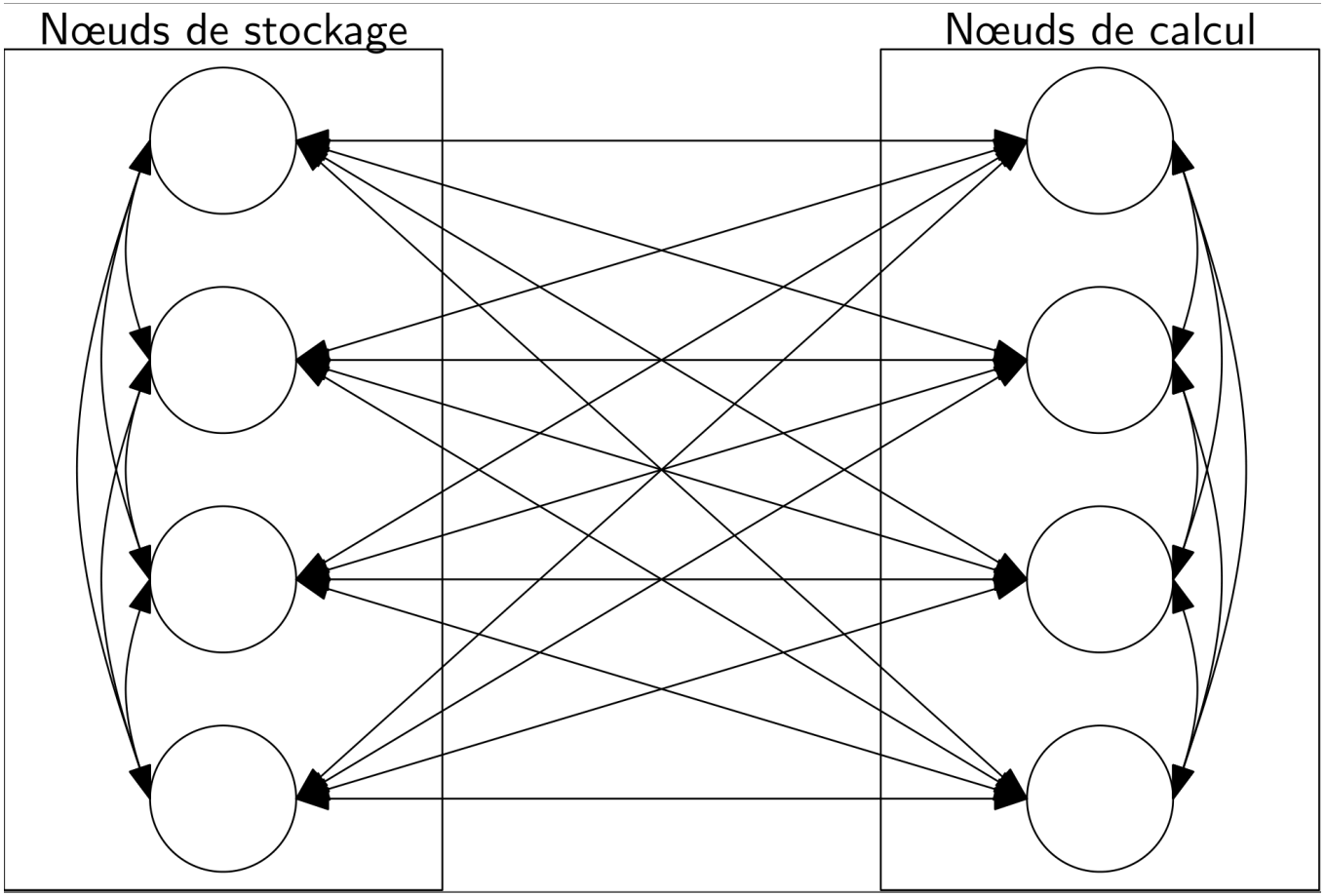
Simulation numérique

- Par exemple: météo, climat
- Des équations différentielles
- Des conditions aux limites
- Décomposition en éléments finis

Calcul distribué

- Beaucoup de calculs
- Relativement peu de transferts

Architecture physique



- Tout le monde peut parler à tout le monde

Architecture logique

Cas général

- Tout le monde parle à tout le monde

Souvent

- Nœuds de stockage: pas de communication
- Nœuds de calcul: communication seulement avec les voisins

Avantage

- Modèle de calcul générique

Difficile à programmer

Sûreté du calcul

- Éviter deux écritures simultanées au même endroit
- Garantir qu'une donnée a été écrite complètement avant d'essayer de la lire

Données

- Minimiser les transferts
- Données loin du calcul

Problème fondamental

- La taille des données augmente plus vite que la vitesse des transfert

Simplification

Limiter la puissance du modèle

- Seulement certains types de programmes parallèles

Rapprocher les données

- Données stockées sur les nœuds de calcul
- Chaque nœud travaille sur ses données

MapReduce

Map

- Fonction appelée à toutes les entrées du jeu de donnée

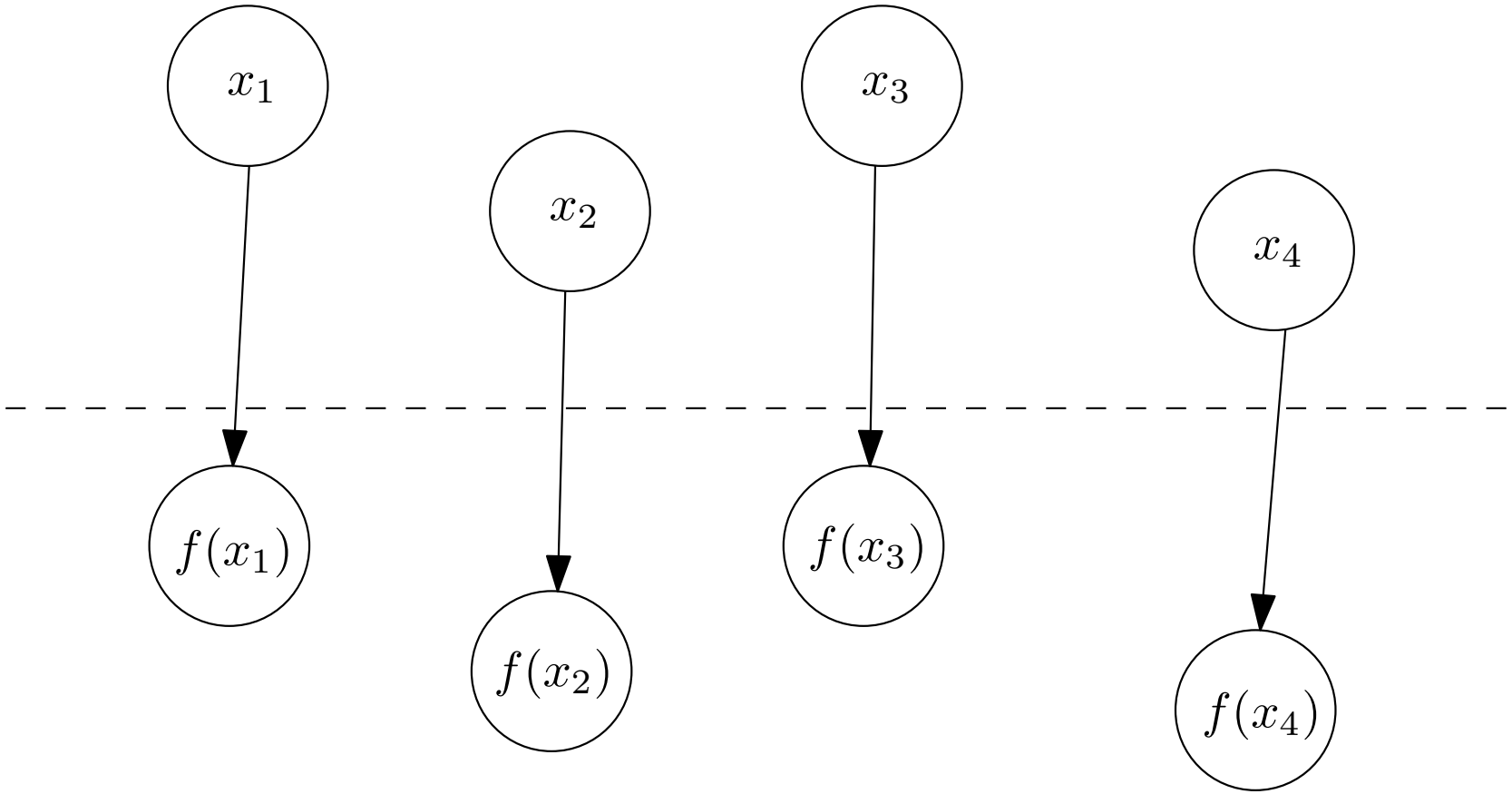
Reduce

- Calcul d'une valeur globale

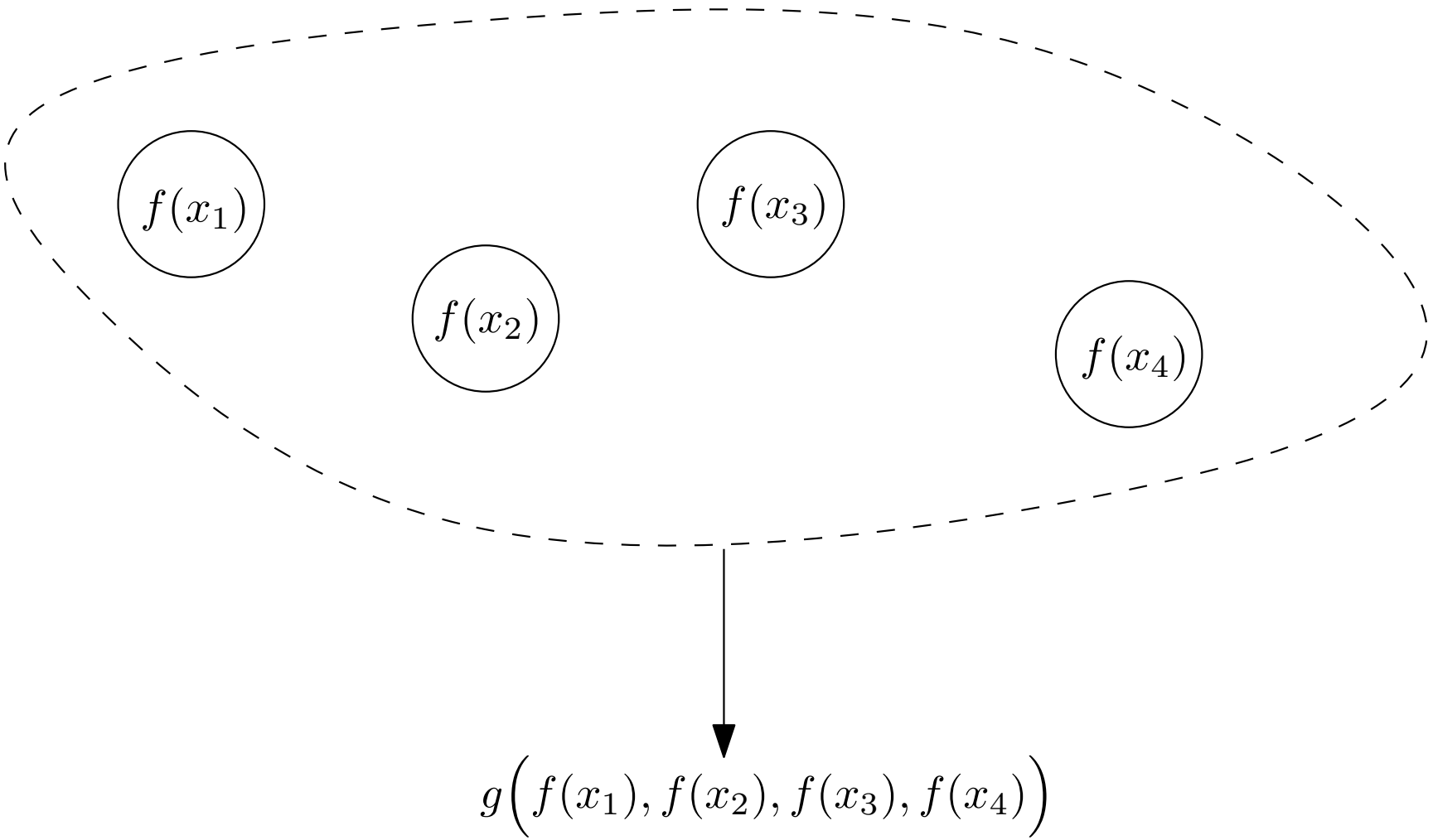
Plus simple

- Seulement deux opérations à écrire
- Pas besoin de connaissances en parallélisme

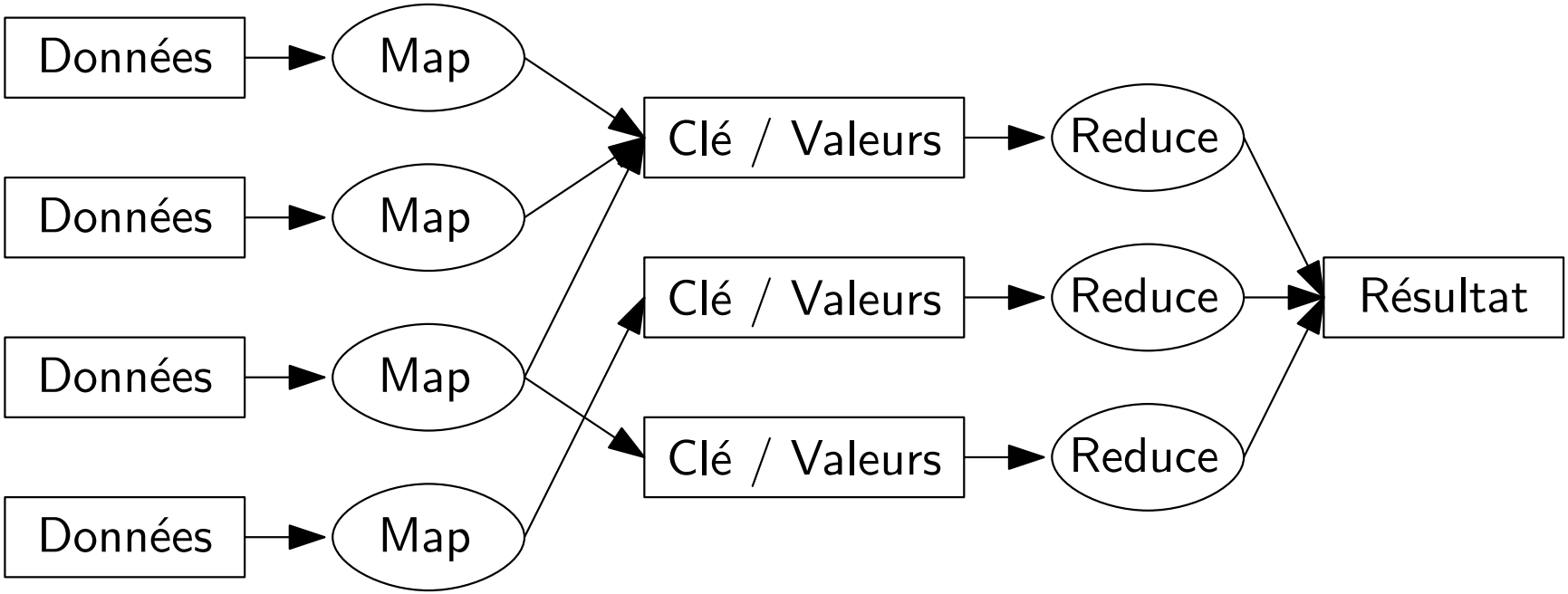
Map



Reduce



MapReduce complet



Avantages

Robustesse: réplication des données

- Même information sur plusieurs nœuds
- Pas de perte en cas de panne
- Tolérance à un certain nombre de défaillances

Transfert: localité

- Traitement des données locales
- Copies pendant la réduction

MapReduce

Facile

- Programmation parallèle plus facile et plus sûre

Mais...

- Besoin d'une version MapReduce des méthodes
- Traitement par lots (*batch processing*)

Souvent

- Besoin d'un enchaînement de MapReduce
- Écriture sur le disque des résultats intermédiaires

Retour vers le passé

Chacun fait son système de parallélisme dans son coin.

Retour vers le futur

Généraliser MapReduce

- Plus de programmes
- Plus de styles de programmation

Travailler en mémoire

- Éviter les écritures sur le disque
- Stocker tous les résultats intermédiaires dans la mémoire

Spark

Traitements

- Par lots
- Interactifs
- Temps réel
- Stockage distribué

Haut niveau

- Moins besoin de se forcer à passer en MapReduce
- Plus de constructions

Principes

Description d'un calcul

- Enchaînement de *transformations*
- Résultats intermédiaires en mémoire

À la fin

- Évaluation de tout le bloc de calcul
- En parallèle
- Stockage permanent du résultat

Resilient Distributed Datasets (RDDs)

Collection d'éléments

- Stockée de façon distribuée
- Calculée et recalculée à la demande, en fonction des besoins

Sources

- Données chargées depuis le HDFS
- Résultat intermédiaire d'un calcul

Propriété

- En mémoire
- Cache sur le disque possible
- Immutable
- Parallélisé

Transformations et actions

Transformation

- RRD $\rightarrow$ RRD
- Pas calculé immédiatement

Action

- RRD $\rightarrow$ résultat concret
- Déclenche le calcul

Description d'un calcul

Séquence d'opérations

- Optmisation à la volée
- Recalcul si nécessaire

Exemples de tranformations

`map(fonction)`	Applique une fonction sur un RRD
`filter(fonction)`	Filtre un RRD
`flatMap(fonction)`	Applique une fonction et aplatit le résultat
`sample(entier)`	Échantillonage aléatoire
`union(rrd)`	Fusion de deux RRD
`groupByKey()`	Regroupe les valeurs associées à la même clé
`reduceByKey(fonction)`	Réduction clé par clé
`sortByKey()`	Tri
`join(rrd)`	$((k, v),\ (k, w)) \rightarrow (k, (v, w))$
`cartesian(rrd)`	Produit cartésien

Exemples d'actions

`reduce(fonction)`	Réduction globale
`collect()`	Récupère une valeur concrète
`count(entier)`	Comptage
`first()`	Premier élément
`take(entier)`	*n* premiers éléments
`saveAsTextFile(fichier)`	Écriture
`foreach(fonction)`	Application d'une fonction

Autres opérations

Cache

- En mémoire, sur disque
- Pour réutiliser des RRD

Bases de données

- Pseudo-relationnel
- Pseudo-SQL

WordCount

```
>>> data = sc.textFile("README.md")
>>> wc = data.flatMap(lambda x: x.split())
          .map(lambda x: (x, 1))
          .reduceByKey(lambda a, b: a + b)
>>> wc
```

Que contient `wc` ?

Pas encore de calcul

```
>>> wc
PythonRDD[36] at RDD at PythonRDD.scala:43
```

Calcul abstrait

- `wc` est un RRD
- La description d'un calcul
- Des données qu'on peut transformer

Calcul effectif

```
>>> result = wc.collect()
16/02/17 14:32:31 INFO SparkContext: Starting job: collect at <stdin>:1
16/02/17 14:32:31 INFO DAGScheduler: Registering RDD 33 (reduceByKey at
16/02/17 14:32:31 INFO DAGScheduler: Got job 8 (collect at <stdin>:1) w
16/02/17 14:32:31 INFO DAGScheduler: Final stage: ResultStage 13 (colle
...
```

Résultat

```
>>> result
[(u'', 67), (u'when', 1) ...]
```

Apprentissage avec MLLib

Tâches

- Classification (SVM, régression logistique, forêts aléatoires,...)
- Régréssion
- Non-supervisé (k-means)
- Recommandation
- Réduction de dimension (PCA, SVD)

Parallèlisme

- Seulement les méthodes adaptatées pour un cluster.

Exemple: classification de spams

Quelques `import`

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import NaiveBayes
```

Exemple: classification de spams

Chargement des données et découpage

```
spambase = sc.textFile("spambase.data")
             .map(lambda line: line.split(","))
training, test = spambase.randomSplit([0.6, 0.4])
```

Exemple: classification de spams

Mise en forme: train

```
training_labels = training.map(lambda line: int(line[-1]))
training_words  = training.map(
                    lambda line: map(
                      lambda x: float(x), line[0:48]))
training_points = training_labels.zip(training_words)
                  .map(lambda pair:
                         LabeledPoint(pair[0], pair[1]))
```

Exemple: classification de spams

Mise en forme: test

```
test_labels = test.map(lambda line: int(line[-1]))
test_words  = test.map(
                lambda line: map(
                  lambda x: float(x), line[0:48]))
test_points = test_labels.zip(test_words)
              .map(lambda pair:
                     LabeledPoint(pair[0], pair[1]))
```

Exemple: classification de spams

Entraînement et évaluation

```
model = NaiveBayes.train(training_points, 1.0)

predictions = test_points.map(
                lambda p: model.predict(p.features))
predictions_labels = predictions.zip(test_labels)
accuracy = float(predictions_labels.filter(
              lambda (x, v): x == v).count())
         / float(test.count())
```