

Ingénierie de la fouille et de la
visualisation de données massives
(RCP216)
Introduction

Michel Crucianu

(prenom.nom@cnam.fr)

<http://cedric.cnam.fr/vertigo/Cours/RCP216/>

Département Informatique
Conservatoire National des Arts & Métiers, Paris, France

1 septembre 2025

Plan du cours

- 2 **Sujet du cours**
- 3 Objectifs et contenu de l'enseignement
- 4 Organisation de l'enseignement
- 5 Quelques références bibliographiques
- 6 Passage à l'échelle de la fouille de données
 - MapReduce et la fouille de données
 - Spark et la fouille de données

Le sujet

- Fouille de données : “*Process of nontrivial extraction of implicit, previously unknown and potentially useful information (such as knowledge rules, constraints, regularities) from data in databases*” (G. Piatetski-Shapiro)
- Données massives : volume peu compatible avec un traitement centralisé
 - Autres caractéristiques : variété (données peu structurées, hétérogènes), vitesse (flux à traiter en temps réel)
 - En général à « faible densité en information »
- Si l'échelle est souvent le problème dominant, les autres caractéristiques posent aussi des difficultés significatives
- Échelle \Rightarrow nécessité de faire « passer à l'échelle » les méthodes de fouille de données
- Faible densité en information \Rightarrow rôle renforcé de la visualisation, fouille interactive

Plan du cours

- 2 Sujet du cours
- 3 Objectifs et contenu de l'enseignement**
- 4 Organisation de l'enseignement
- 5 Quelques références bibliographiques
- 6 Passage à l'échelle de la fouille de données
 - MapReduce et la fouille de données
 - Spark et la fouille de données

Objectifs de l'enseignement

- Fouille de données massives :
 - Faire comprendre les approches de passage à l'échelle
 - Présenter les méthodes de résolution de certains problèmes fréquents
 - Familiariser avec des logiciels permettant l'exécution distribuée
 - Faire appliquer ces méthodes sur des données réelles
- Visualisation et interaction :
 - Faire comprendre les principes et les enjeux
 - Présenter des techniques usuelles et leurs utilisations
 - Familiariser avec la programmation de ces techniques

⇒ Capacité à mettre œuvre, sur des données massives, des techniques de fouille de données et de visualisation interactive

Contenu du cours (15 séances)

- Approches du passage à l'échelle : réduction de la complexité, distribution
- Analyse de données et réduction de dimension - ACP, AFD, ACM
- Recherche par similarité. Systèmes de recommandation
- Classification automatique
- Fouille de textes
- Apprentissage supervisé à large échelle
- Introduction aux graphes et réseaux sociaux
- Graphes : centralités, modèles, détection de communautés, dynamique
- Introduction à la visualisation d'information
- Visualisation de graphes
- Fouille de séquences et de flux de données
- Aspects éthiques dans la fouille de données massives

Contenu des travaux pratiques

- Introduction à Python et à Spark
- Manipulation de données numériques
- Échantillonnage
- Analyse en composantes principales
- Classification automatique
- Fouille de données textuelles
- SVM linéaires
- Fouille de graphes et réseaux sociaux
- Introduction à la visualisation
- Visualisation de graphes
- Traitement de flux de données
- Détection et réduction des disparités

Plan du cours

- 2 Sujet du cours
- 3 Objectifs et contenu de l'enseignement
- 4 Organisation de l'enseignement**
- 5 Quelques références bibliographiques
- 6 Passage à l'échelle de la fouille de données
 - MapReduce et la fouille de données
 - Spark et la fouille de données

Prérequis

- En mathématiques : bonnes connaissances mathématiques et statistiques générales, maîtrise de méthodes statistiques pour la fouille de données
- En informatique : connaissances élémentaires en programmation, connaissance de techniques de gestion de données massives faiblement structurées, de passage à l'échelle par distribution
- Questionnaire d'auto-évaluation anonyme :
<http://cedric.cnam.fr/vertigo/Cours/RCP216/questionnaire.html>

Dans le cadre du certificat de spécialisation « Analyste de données massives », ces prérequis se traduisent, en partie, par l'exigence d'avoir suivi NFE204 et STA211 avant de commencer RCP216

Organisation

- Enseignant[s] : Michel Crucianu, [Raphaël Fournier-S'niehotta]
- S1 : formation ouverte à distance (FOAD)
 - Exclusivement en ligne sur <http://lecnam.net>
 - Supports écrits et/ou vidéo
 - Travaux pratiques à distance sur le JupyterHub du Cnam
- Supports écrits : <http://cedric.cnam.fr/vertigo/Cours/RCP216/>
 - Cours : transparents en ligne pour tout le cours, explications détaillées en HTML, vidéos sur Moodle (<http://lecnam.net>) pour des ajouts récents
 - Travaux pratiques : contenu détaillé en HTML, corrigés mis en ligne env. 2 semaines après la séance
- Forum informel sur Moodle (<http://lecnam.net>, page de RCP216)

5h30-19h30

9h45-21h45

Evaluation

■ Examen sur table

- S1 : examen en février avec rattrapage en avril

n septembre

- Planification des examens : <http://www.cnam-paris.fr/suivre-ma-scolarite/>
rubrique Examens
- Les notes **ne sont pas** transmises par téléphone ou courriel (règle Cnam)

■ Projet : appliquer des techniques de fouille (**avec Spark**) et visualisation sur des données réelles

- Sujets proposés par auditeurs ou enseignants, env. 2 mois après le démarrage du cours
- 2 dates limite chaque semestre, synchronisées avec l'examen principal et respectivement l'examen de rattrapage

■ Note finale = moyenne (non pondérée) entre note de projet et note d'examen

Plan du cours

- 2 Sujet du cours
- 3 Objectifs et contenu de l'enseignement
- 4 Organisation de l'enseignement
- 5 Quelques références bibliographiques**
- 6 Passage à l'échelle de la fouille de données
 - MapReduce et la fouille de données
 - Spark et la fouille de données

Références bibliographiques

- Fouille de données massives
 - J. Leskovec, A. Rajaraman, J. Ullman. *Mining of Massive Datasets*. Cambridge University Press. <http://www.mmms.org>
 - Karau, H., A. Konwinski, P. Wendell et M. Zaharia. *Learning Spark - Lightning-Fast Big Data Analysis*, O'Reilly.
 - Ryza, S., U. Laserson, S. Owen and J. Wills. *Advanced Analytics with Spark - Patterns for Learning from Data at Scale*, O'Reilly.
- Visualisation des données
 - B. Fry. *Visualizing data*. O'Reilly, 2008.
 - R. Spence. *Information Visualization, design for interaction*. Prentice Hall, 2007.
- sans oublier, entre autres
 - C. O'Neil, *Weapons of Math Destruction*. Crown, 2016.
 - Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning Limitations and Opportunities*. <https://fairmlbook.org>

Plan du cours

- 2 Sujet du cours
- 3 Objectifs et contenu de l'enseignement
- 4 Organisation de l'enseignement
- 5 Quelques références bibliographiques
- 6 Passage à l'échelle de la fouille de données
 - MapReduce et la fouille de données
 - Spark et la fouille de données

Le passage à l'échelle

- vise notamment la capacité à traiter de très gros volumes de données
- Faible densité d'information \Rightarrow il n'est pas toujours souhaitable d'exclure *a priori* certaines composantes des données
 - Travailler sur un échantillon trop petit : les « régularités » recherchées ne se manifestent pas suffisamment pour être détectables...
 - Travailler sur un nombre de variables trop faible : les « régularités » sont incomplètes, les capacités prédictives sont insuffisantes...
- Peut concerner d'autres aspects : hétérogénéité, nombre de variables, etc.

Le passage à l'échelle : approches complémentaires

- Réduction de la complexité (N : volume des données)
 - Sans changer l'ordre de complexité : réduction de dimension, calculs sur échantillon
 - Réduction de l'ordre de complexité : $O(N) \rightarrow O(\log(N))$ ou $O(cst.)$,
 $O(N^2) \rightarrow O(N \log(N))$
 - Méthodes exactes *ou approximatives*!

- Parallélisation des calculs
 - Architectures parallèles dédiées
 - Rapport coût / performance très élevé (architectures et technologies dédiées)
 - Goulot d'étranglement entre stockage de masse et unités centrales
 - Architecture parallèles intégrées (GPU et processeurs vectoriels)
 - Dédiées au calcul matriciel
 - Capables de traiter des volumes de données moyens (dizaines Go) mais croissant
 - Distribution des calculs sur un grand nombre d'ordinateurs standard (coût bas), avec
 - Minimisation des échanges avec le stockage de masse, tout en assurant la fiabilité aux pannes
 - Minimisation des échanges de données entre ordinateurs
 - Capables de traiter des données massives (To à Po)

Problématique de l'exécution distribuée

- Comment faciliter l'implémentation d'une grande diversité d'algorithmes sur une plateforme distribuée ?
 - Mécanisme d'exécution composé d'opérations simples et génériques
- Comment réduire la fragilité aux pannes, inévitables pour des calculs longs réalisés sur des plateformes comportant de grands nombres d'ordinateurs ?
 - Mécanisme de **réplication** pour le stockage fiable des données
 - **Partitionnement** des calculs en « grains » traçables qui peuvent être (ré)affectés à d'autres ordinateurs

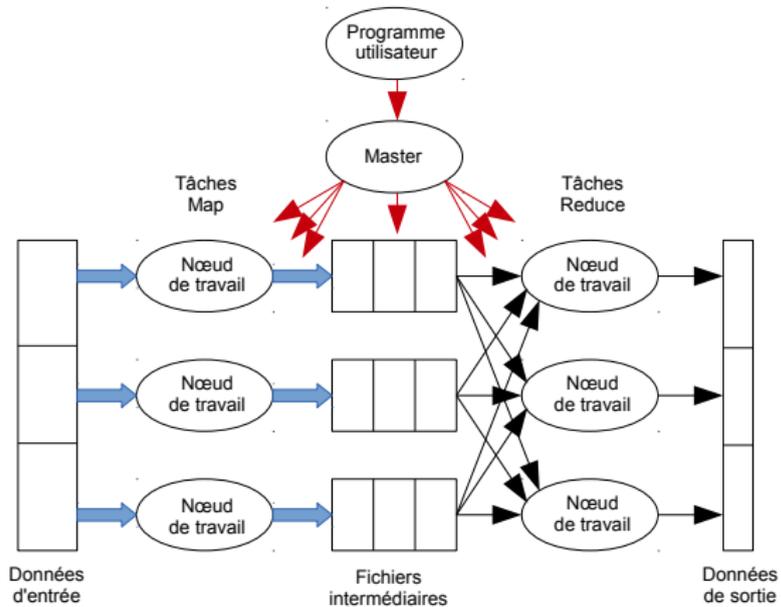
MapReduce

- Mécanisme d'exécution simple
 - Permettant un parallélisme massif
 - Tolérant aux pannes
- Principe : décomposer le calcul en deux types de tâches élémentaires et uniformes, *Map* (exécutées en premier) et *Reduce* (exécutées ensuite), qui peuvent être facilement assignées à des nœuds de calcul
- Autre idée de MapReduce : si les données sont (beaucoup) plus volumineuses que les programmes, stocker localement les données et transférer plutôt les programmes pour les rapprocher des données

MapReduce : fonctionnement général

- 1 L'ensemble de données à traiter est découpé en *fragments* (*chunks*)
 - 2 Chaque tâche *Map* est assignée à un nœud de calcul qui reçoit un ou plusieurs fragments que la tâche *Map* transforme en une séquence de paires (clé, valeur)
 - 3 Chaque tâche *Reduce* est associée à une ou plusieurs clés et est assignée à un nœud de calcul
 - 4 Les paires (clé, valeur) produites par les *Map* sont groupées par clés et stockées sur les nœuds de calcul qui exécuteront les tâches *Reduce* respectives (étape *shuffle*)
 - 5 Chaque tâche *Reduce* combine, pour chaque clé qui lui est associée, les valeurs des paires (clé, valeur) avec cette clé ; les résultats sont stockés et constituent le résultat du traitement
- Le programmeur écrit les fonctions *Map* et *Reduce*, le *framework* se charge du reste
 - Hadoop : implémentation *open source* de *MapReduce*

MapReduce : exécution d'un programme



MapReduce : un exemple classique

Tâche à réaliser : compter le nombre d'occurrences de chaque mot dans une collection de documents textuels

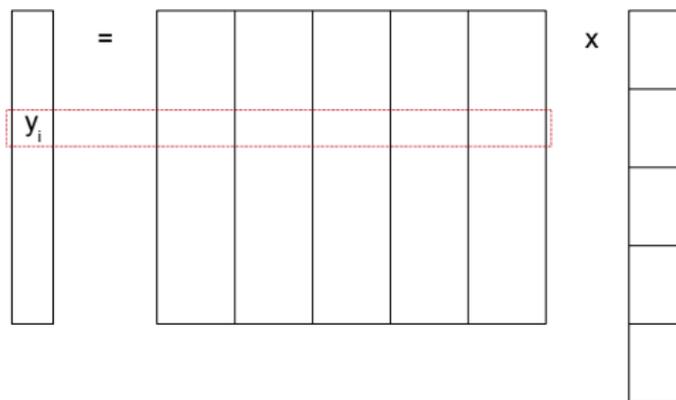
- 1 On considère, pour simplicité, que chaque document constitue un fragment
- 2 Chaque tâche *Map* reçoit un fragment, le découpe en mots et, pour chaque occurrence d'un mot, génère une paire [mot, 1] (clé := mot, valeur := 1)
- 3 Les paires [mot, 1] sont groupées par mots et stockées
- 4 Chaque tâche *Reduce* reçoit les paires correspondant à un mot, fait l'addition des valeurs qui lui sont associées et stocke les résultats

Remarque : l'addition étant associative et commutative, il est possible de transférer dans les *Map* une partie des calculs des *Reduce*; ainsi, une tâche *Map* peut aussi additionner le nombre d'occurrences de chaque mot dans le fragment qu'elle reçoit et générer plutôt des paires [mot, nombre d'occurrences du mot dans le fragment]. Cela permet de minimiser les échanges de données.

MapReduce : un autre exemple classique

Tâche à réaliser : multiplication (grande) matrice \times vecteur

$$\mathbf{y} = \mathbf{M} \times \mathbf{x} \quad \text{ou} \quad y_i = \sum_{j=1}^n m_{ij} x_j$$

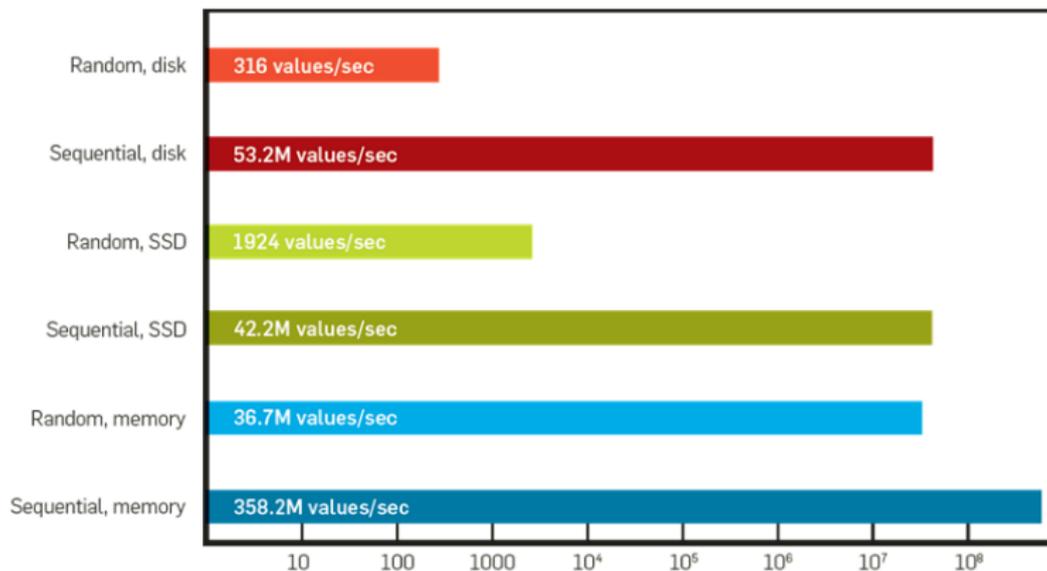


MapReduce : multiplication matrice \times vecteur

- 1 La très grande matrice \mathbf{M} est découpée en fragments qui sont des groupes de colonnes ; un fragment doit tenir dans la mémoire d'un nœud de calcul
- 2 Le vecteur \mathbf{x} est aussi découpé en groupes de lignes de façon correspondante
- 3 Chaque tâche *Map* reçoit un fragment de la matrice \mathbf{M} et le fragment correspondant de \mathbf{x} ; pour chaque élément m_{ij} de son fragment, calcule $m_{ij}x_j$ et génère $(i, m_{ij}x_j)$
- 4 Les paires $(i, m_{ij}x_j)$ sont groupées par i et stockées
- 5 Chaque tâche *Reduce* reçoit les paires correspondant une valeur de la clé i , fait l'addition des valeurs qui lui sont associées et stocke les résultats

Remarque : toujours grâce aux propriétés de l'addition, il est possible de transférer dans les *Map* une partie des calculs des *Reduce* pour minimiser les échanges de données ; ainsi, une tâche *Map* peut aussi additionner les $m_{ij}x_j$ pour tous les j de son groupe de colonnes.

Hiérarchie de stockage



* Disk tests were carried out on a freshly booted machine (a Windows 2003 server with 64GB RAM and eight 15,000RPM SAS disks in RAID5 configuration) to eliminate the effect of operating-system disk caching. SSD test used a latest generation Intel high-performance SATA SSD.

(figure issue de *Communications of the ACM* 2009 (8) : 36-44)

MapReduce et la fouille de données

- Un parallélisme massif serait inenvisageable sans un mécanisme robuste de reprise sur panne
- Le mécanisme de reprise sur panne de *MapReduce* impose le stockage des résultats des *Reduce*
- Pour beaucoup d'algorithmes—dont certains de fouille de données—ces résultats sont intermédiaires : l'itération suivante doit prendre ces résultats comme données d'entrée
- Le stockage des résultats intermédiaires, imposé par le mécanisme de reprise sur panne, engendre un ralentissement très significatif de ces algorithmes

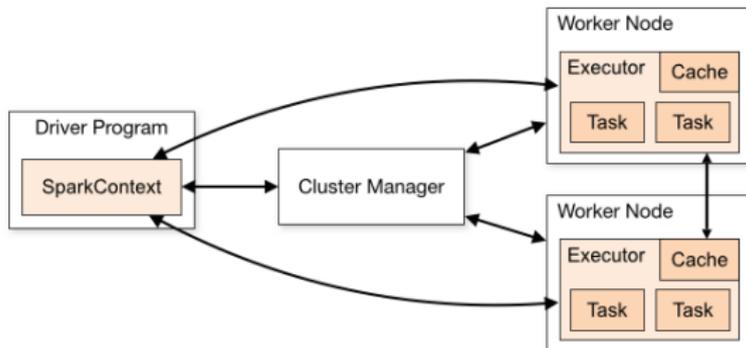
La solution de Spark

- <https://spark.apache.org>
 - Revoir la présentation de Spark dans NFE204 :
<http://b3d.bdpedia.fr/spark-batch.html>
(rappel MapReduce : si les données sont beaucoup plus volumineuses que les programmes, stocker localement les données et transférer plutôt les programmes)
 - Idée clé de Spark : si les données intermédiaires sont volumineuses, les garder en mémoire vive ; en cas de panne *recalculer ce qui a été perdu*
 - Concept important : *Resilient Distributed Dataset* (RDD) = collection de données partitionnée en fragments (*partitions, chunks*) distribués sur différents nœuds de calcul et conservés en mémoire
- Fort impact sur la **réduction de la latence** des algorithmes itératifs
- A partir de la *v1.6* introduction des *DataFrame*, collections de données distribuées avec des colonnes nommées, conceptuellement similaires aux Data Frames de R

Spark : bibliothèques existantes

- Bibliothèques importantes pour nous (utilisées dans les TP de RCP216) :
 - MLlib : analyse des données, fouille de données, apprentissage statistique
 - GraphX : calculs sur les graphes (*data-parallel* et *graph-parallel*)
 - Spark Streaming : traitement de flux de données (par ex. Twitter, ZeroMQ, Kinesis, *sockets* TCP)
- Spark est écrit en Scala et peut être utilisé à partir de Scala, Java, Python et R (dans une moindre mesure, via *sparklyr* ou *SparkR* – *dépréciée à partir de v4.0.0*)
- Les programmes en Scala tournent dans une machine virtuelle Java (JVM), des programmes en Scala peuvent appeler des bibliothèques en Java et inversement
- *Deep learning* : TorchDistributor pour exécution PyTorch et Lightning, Horovod pour exécution TensorFlow

Spark : déploiement



(figure issue de <https://spark.apache.org>)

■ Spécificités :

- Gestionnaires de cluster : propre à Spark (mode *standalone*) ou Apache Mesos ou Hadoop YARN ou Kubernetes
- Isolation entre applications Spark différentes : chaque application tourne dans une JVM et est donc isolée des autres, peut communiquer avec d'autres uniquement à travers des fichiers
- Pour des raisons d'efficacité, le *driver* devrait tourner sur un nœud du cluster

Autres plate-formes d'exécution parallèle : Dask et Ray

- Dask (<https://www.dask.org/>) : native Python
 - Objectif : parallélisation facile de programmes qui emploient les API Python courantes comme NumPy et Pandas
 - Support GPU via RAPIDS cuDF and CuPy
 - Pauvreté des opérations relationnelles
- Ray (<https://www.ray.io/>) : API Python mais *core* C++
 - Objectif : parallélisation générale d'algorithmes en Python
 - Bibliothèques spécifiques disponibles pour l'apprentissage statistique (supervisé, par renforcement)
 - Primitives : Task (fonction *stateless*), Actor (instance, *stateful*), Object (donnée)
 - Pauvreté des opérations relationnelles

→ Choix entre Spark, Dask et Ray ?

- Adéquation à la tâche à évaluer attentivement, d'autant plus qu'on peut aussi envisager Spark on Ray, Ray on Spark, Dask on Ray...
- Données massives : Spark pour préparation des données et application d'un modèle, Ray pour développement du modèle
- Pour une comparaison plus détaillée voir par ex. [cette discussion](#)