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

31 août 2026

Plan du cours

Le sujet

- Fouille de données : "*Processus d'extraction non triviale d'informations implicites, inconnues auparavant et potentiellement utiles (sous forme de règles, contraintes, régularités) à partir de données issues de bases de données*" (Gregory Piatetsky-Shapiro)
- Ce cours n'est pas une introduction à la fouille de données en général
  - On suppose connues les bases de l'apprentissage statistique supervisé et non supervisé
  - On suppose qu'un enseignement sur les bases de données documentaires et distribuées a été suivi

→ Notre sujet : le passage à l'échelle de la fouille de données
  - Comment traiter des volumes qui dépassent les capacités d'une seule machine ?
  - Comment tirer parti d'architectures multi-cœur ou distribuées ?
  - Comment adapter les algorithmes aux contraintes des données massives ?

Plan du cours

Les 3V des données massives

- Rapport de 2001 du META Group (actuellement Gartner) : cadre de référence pour caractériser les données massives, les 3V
  - **Volume** : augmentation des quantités de données stockées et traitées, souvent bien plus rapide que la progression des capacités des serveurs individuels
  - **Variété** : données non structurées ou semi-structurées (XML, JSON, textes, images, vidéo, audio, capteurs), exigeant des méthodes de prétraitement et de fouille adaptées
  - **Vélocité** : arrivée en flux continu, traitement en temps réel ou à faible latence (détection de fraude, pilotage industriel, recommandation en ligne)

Volume : quelques ordres de grandeur

- Un moteur d'avion de ligne génère environ 1 To de données par heure de vol (capteurs de vibration, température, pression, etc.)
- Le télescope spatial Rubin Observatory produit environ 20 To de données astronomiques par nuit d'observation
- Les grandes plateformes de *streaming* vidéo ingèrent plusieurs centaines de Po de contenu au total
- À l'échelle mondiale, le volume de données créées et consommées a dépassé 100 Zo (zettaoctets) en 2023 (estimations IDC)

Au-delà des 3V

- D'autres caractéristiques s'ajoutent souvent aux 3V mentionnés
  - **Véracité** : qualité et fiabilité des données, bruit, erreurs, données manquantes
  - **Valeur** : retour sur investissement attendu du traitement
  - **Visibilité** : capacité à extraire des connaissances exploitables

→ Caractéristique importante : la **faible densité en information**

- L'information utile est « diluée » dans un très grand nombre d'observations et de variables
- Un échantillon trop petit ou une sélection trop restreinte de variables a peu de chances de donner de bons résultats

→ difficile de réduire le volume (nombre d'observations de variables) sans perdre en **valeur/visibilité**

→ rôle renforcé de la visualisation, fouille interactive

Domaines d'application (1)

- Commerce en ligne et recommandation
  - Amazon, Netflix : traces d'interaction (clics, achats, évaluations) de centaines de millions d'utilisateurs

  → Filtrage collaboratif à large échelle pour personnaliser les recommandations en temps quasi-réel

- Santé et bioinformatique
  - Séquençage génomique : données de très haute dimension (millions de positions sur le génome)
  - Cohortes de patients : millions de dossiers médicaux, mélange de données textuelles et structurées

  → Réduction de dimension, encodage de texte, apprentissage à grande échelle

Domaines d'application (2)

- Réseaux sociaux et graphes du web
  - Facebook, X, LinkedIn : graphes de plusieurs milliards de nœuds (utilisateurs), dizaines de milliards d'arêtes (liens)

  → Détection de communautés, calcul de centralités, propagation d'influence

- Villes intelligentes et IoT
  - Réseaux de capteurs urbains (trafic, qualité de l'air, énergie) : flux continus à traiter en temps réel

- Traitement automatique des langues à grande échelle
  - Entraînement des LLM : corpus de plusieurs centaines de milliards de tokens, milliers de GPU pendant des semaines
  - Défis de passage à l'échelle même en *inférence*

Plan du cours

Le passage à l'échelle

- Nous nous intéressons aux trois défis mentionnés – volume, variété, vélocité – en insistant sur le premier
- Nous étudions les opérations de fouille en environnement distribué et certains problèmes fréquents en fouille de données massives
- Nous examinons le rôle de la visualisation et de l'interaction, non seulement pour la présentation des résultats mais aussi lors des opérations de fouille

→ « Passage à l'échelle » (*scalability*) = ici, capacité à faire face à une augmentation forte du volume de données à traiter

- Notion plus large : peut aussi concerner le nombre de variables, l'hétérogénéité des données, etc.

Plan du cours

Deux grandes familles d'approches

- Face à un volume de données qui dépasse les capacités d'une architecture centralisée classique, deux grandes familles d'approches sont possibles
  - **Réduire** le travail à faire, sans distribuer les calculs
  - **Distribuer** les données et les calculs sur un grand nombre d'unités de traitement

→ Ces deux familles ne sont pas exclusives et peuvent être combinées

Approches sans distribution

1. **Réduction du volume de données** : travailler sur un sous-ensemble représentatif pour faire tenir les données sur une seule machine et réduire volume calculs
   - → Échantillonnage : réduit le nombre d'observations
   - → Réduction de dimension : diminue le nombre de variables
     - Risque : perte excessive d'information si faible densité en information

2. **Réduction de l'ordre de complexité algorithmique** : travailler sur **toutes** les données, exploiter leurs propriétés de similarité pour réduire l'ordre de complexité
   - Ex. : $O(N) \rightarrow O(\log N)$ pour recherche des plus proches voisins avec structure d'index
   - → Hachage sensible à la similarité (*Locality Sensitive Hashing*, LSH)
   - → Bases de données vectorielles (*vector databases*)
     - Risque : peu efficaces si dimension très élevée (malédiction de la dimension)

Approches sans distribution : calcul efficace mono-machine

3. **Exploitation d'autres solutions d'amélioration de l'efficacité** : exploiter le parallélisme multi-cœur, minimiser les recopies mémoire, etc.
   - Polars : bibliothèque *DataFrame* écrite en Rust, format en colonnes (Apache Arrow), parallélisation automatique sur tous les cœurs
     - 5 à 10 fois plus rapide que Pandas pour filtrage, agrégation, jointure
   - Dask : solution Python native, étend les API NumPy/Pandas à des données qui ne tiennent pas en mémoire vive, exploite efficacement le stockage de masse
     - Découpe les calculs en tâches exécutées en parallèle (multi-cœurs, mais aussi *cluster*)

Approches distribuées

- Lorsque le volume de données est tel qu'une seule machine ne suffit plus (malgré les optimisations vues)

→ Distribuer données et calculs sur un nombre élevé d'ordinateurs standard reliés par un réseau local haut débit (*cluster*)

- Historique de la parallélisation des calculs
  - Architectures parallèles dédiées (Cray, etc.)
    - Rapport coût / performance très élevé (architectures et technologies dédiées)
    - Goulot d'étranglement entre stockage de masse et unités centrales
  - → Architecture parallèles intégrées (GPU, processeurs matriciels)
    - Dédiées au calculs matriciels
    - 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 [1], 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

[1] Soit p la probabilité de panne d'un nœud et n le nombre de nœuds, alors la probabilité d'apparition d'au moins une panne dans le *cluster* est $P = 1 - (1 - p)^n$. Si $p = 0{,}001$ et $n = 10000$ alors $P = 0{,}999 \gg p$.

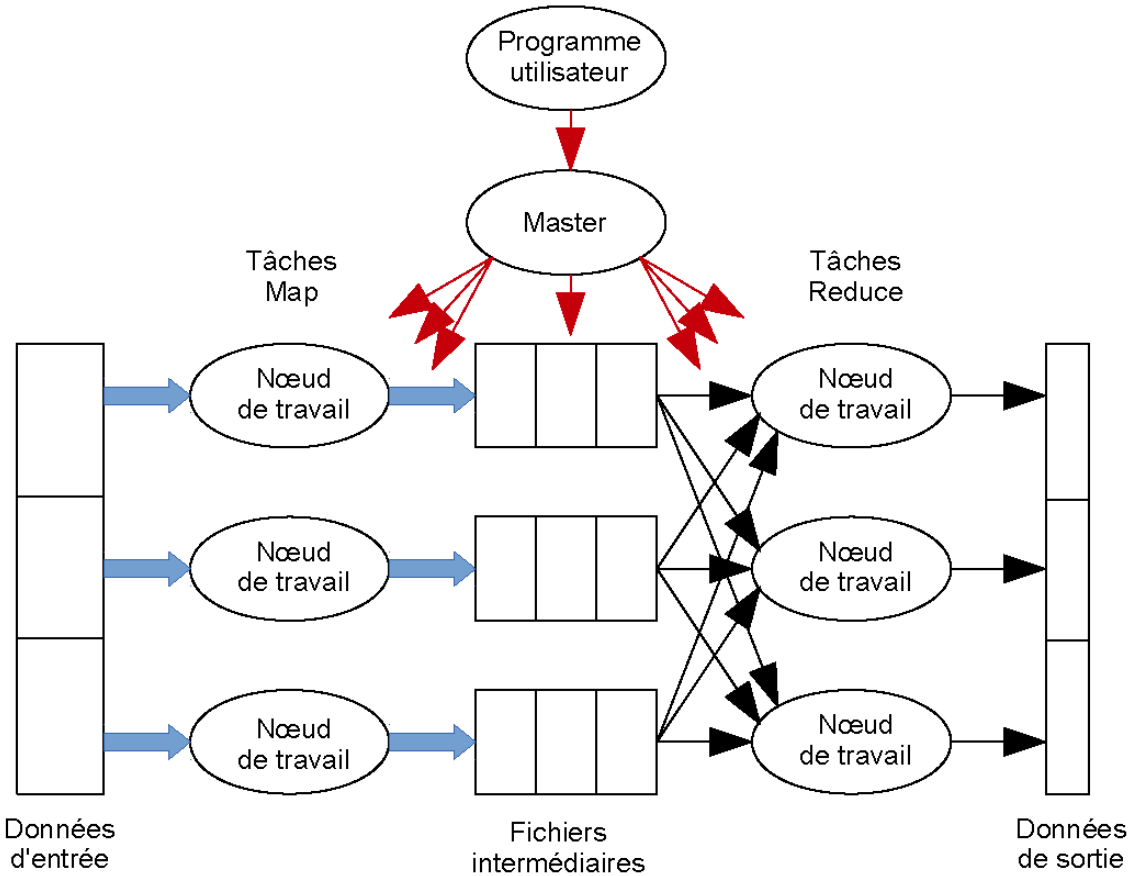
MapReduce : le modèle fondateur

- Modèle popularisé par Google, implémenté dans le *framework open source* Hadoop
- Premier mécanisme d'exécution distribuée à grande échelle

- Décomposition de tout traitement en deux types de tâches élémentaires et uniformes
  - Tâches *Map* : chacune reçoit un fragment de données et produit une séquence de paires [clé, valeur]
  - Tâches *Reduce* : chacune dédiée à une clé donnée, reçoit des *Map* l'ensemble des valeurs associées et les combine pour produire un résultat

- Le programmeur écrit seulement les fonctions *Map* et *Reduce* ; le *framework* distribue les données, affecte les tâches, gère les pannes et collecte les résultats
- Autre idée de *MapReduce* : si les données sont (beaucoup) plus volumineuses que les programmes, stocker localement les données et transférer les programmes pour les rapprocher des données

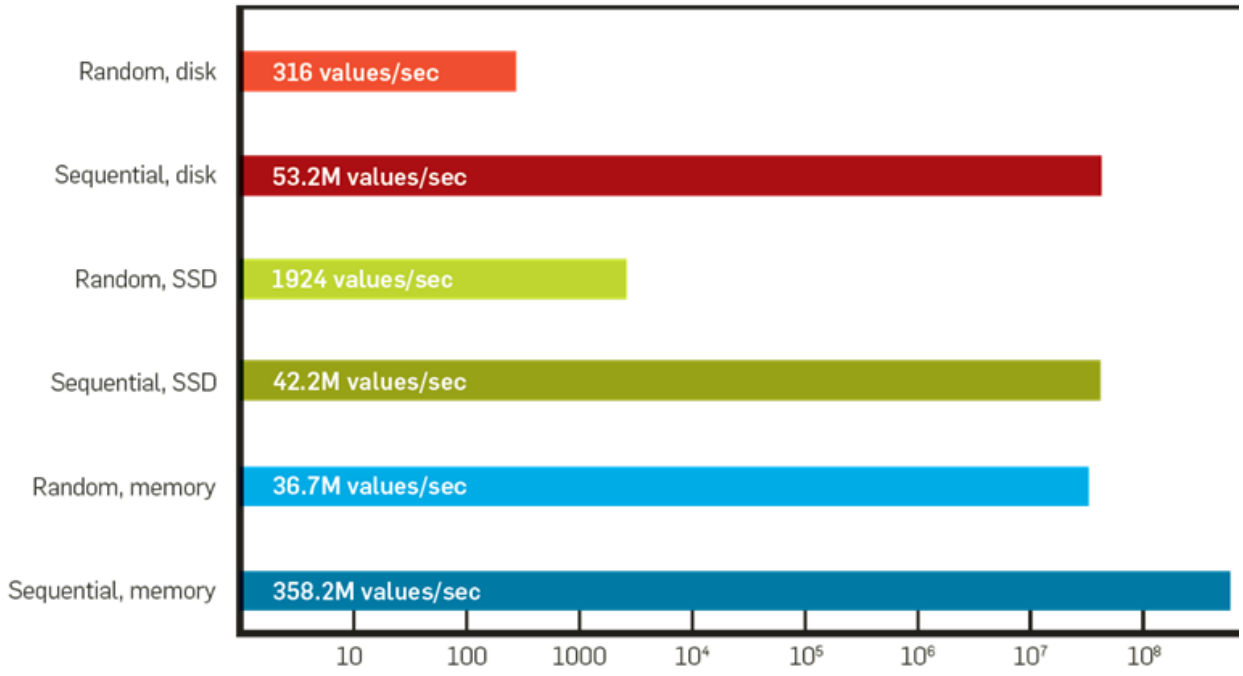
MapReduce : exécution d'un programme



Hiérarchie de stockage

- La mémoire vive est plusieurs ordres de grandeur plus rapide que l'accès au disque

→ au cœur des choix architecturaux de Spark par rapport à *MapReduce*



* 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 : une limite importante

- Limite forte pour les algorithmes itératifs, courants en fouille de données
- Le mécanisme de reprise sur panne impose que les résultats intermédiaires de chaque étape *Reduce* soient écrits sur disque avant de pouvoir être utilisés par l'étape suivante
- Pour un algorithme à k itérations : k aller-retours disque $\Rightarrow$ ralentissement considérable (cf. hiérarchie de stockage)

$\rightarrow$ Une séance entière de ce cours sera consacrée à *MapReduce* et à ses évolutions

Spark : conserver les données en mémoire

- https://spark.apache.org

→ Idée clé : si les données intermédiaires sont volumineuses, les garder en mémoire vive ; conserver aussi l'historique des opérations ayant permis de les produire
  - En cas de panne : recalculer les données perdues à partir des dernières données disponibles, sans reprise depuis le début

- Accélération de 10 à 100 fois par rapport à *MapReduce* classique sur les algorithmes itératifs

- *Resilient Distributed Dataset* (RDD) : collection de données partitionnée, distribuée, conservée autant que possible en mémoire
- API *Dataset/DataFrame* : typage fort, optimiseur de plans d'exécution (*Catalyst*)

Spark : librairies existantes

- **MLlib** : apprentissage statistique, analyse et fouille de données
- **Spark Streaming / Structured Streaming** : traitement de flux de données
- **GraphX / GraphFrames** : calcul sur les graphes (*data-parallel* et *graph-parallel*)
- **SparkNLP** : traitement de la langue

- Spark est écrit en Scala, utilisable depuis Scala, Java, Python (PySpark) et R

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

- Dask : peut aussi être déployé en *cluster*
  - Conserve les API familières de Pandas et NumPy
  - Généralement plus facile à prendre en main que Spark pour les utilisateurs Python, mais moins adapté aux très grandes échelles

- Ray : développé en C++ depuis 2017, objectif de parallélisation générale d'algorithmes Python (utilisé entre autres par OpenAI)
  - Primitives : *Task* (fonction *stateless*), *Actor* (instance, *stateful*), *Object* (donnée)
  - Particulièrement adapté à l'entraînement distribué de modèles prédictifs (y compris apprentissage par renforcement) et au service de modèles en production
  - Support des opérations relationnelles plus limité que Spark

→ Le choix dépend de la taille des données, de la nature des calculs, de la familiarité avec les API, du support disponible

Vue d'ensemble des principales bibliothèques/plateformes

	Pandas	**Polars**	**Dask**	**Spark**	**Ray**	**Hadoop/MR**
Distribution	Non	Non	Optionnelle	Oui	Oui	Oui
Multi-cœur	Non	Oui	Oui	Oui	Oui	Oui
API Python	Oui	Oui	Oui	Oui (PySpark)	Oui	Non natif
Données $>$ RAM	Non	Partiel	Oui	Oui	Oui	Oui
Algo. itératifs	Oui	Oui	Oui	Très efficace	Très efficace	Lent (disque)
ML intégré	Non	Non	Partiel	MLlib	Ray Train	Non
Prise en main	⋆ ⋆ ⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆⋆
Point fort	Référence universelle	Rapidité mono-machine	Pandas $\rightarrow$ distribué	Batch & ML distribués	ML & RL distribué	Robustesse maturité

- Un tableau comparatif plus complet sera présenté à la fin du chapitre sur les approches distribuées
- Domaine en évolution rapide !

Plan du cours

Objectifs de l'enseignement

- Comprendre les approches de passage à l'échelle de la fouille de données
- Découvrir des outils permettant un traitement efficace sans distribution
- Se familiariser avec Spark comme plateforme de référence pour l'exécution distribuée
- Mettre en œuvre des méthodes de fouille sur données textuelles, graphes et flux
- Comprendre les principes et techniques de la visualisation de données et de graphes
- Prendre conscience des enjeux éthiques liés à la fouille de données massives

Contenu du cours

1. Introduction : données massives, approches de passage à l'échelle, vue d'ensemble
2. Approches sans distribution : réduction de volume, calcul efficace (*Polars*, *Dask*)
3. Calcul distribué : MapReduce, Spark (RDD, *DataFrame*, MLlib), Ray
4. Réduction de l'ordre de complexité : LSH, bases de données vectorielles
5. Apprentissage statistique à large échelle : SVM linéaires, *SGD* distribué
6. Données textuelles 1 : traitements classiques, TF-IDF, LSA, plongements lexicaux
7. Données textuelles 2 : *Transformers*, BERT, LLM
8. Graphes 1 : définitions, propriétés, algorithmes fondamentaux
9. Graphes 2 : détection de communautés
10. Visualisation de données : principes, perception, techniques
11. Visualisation de graphes : critères, algorithmes de spatialisation
12. Systèmes de recommandation : filtrage collaboratif, factorisation matricielle, graphes
13. Flux de données : traitement de flux, *Spark Streaming*, *Kafka*
14. Éthique 1 : équité, critères observationnels
15. Éthique 2 : biais, détection et correction de l'iniquité

Contenu des travaux pratiques

- Introduction à Python, NumPy, Matplotlib ; Pandas et Polars
- Introduction à Spark, DataFrames Spark ; comparaison Spark / Dask
- Comparaison SVM, régression logistique, forêt aléatoire, réseau de neurones (Spark)
- Introduction à SparkNLP ; utilisation de GloVe ; comparaison GloVe vs BERT
- Chemins et centralités avec GraphFrames ; fouille du graphe Medline
- Visualisation de données volumineuses (Plotly Dash / Dask) ; Gephi et NetworkX
- Filtrage collaboratif par factorisation matricielle avec Spark
- Traitement de flux avec Spark Streaming
- Évaluation de modèle avec/sans variables protégées ; correction *a posteriori* d'un modèle

Réalisés sur une plateforme JupyterLab du Cnam, avec des données assez peu volumineuses : comprendre les mécanismes plutôt que les mettre en œuvre sur une infrastructure industrielle

Plan du cours

Références bibliographiques

- Fouille de données massives
  - J. Leskovec, A. Rajaraman, J. Ullman. *Mining of Massive Datasets*. Cambridge University Press. http://www.mmds.org
  - Damji, J. S., B. Wenig, T. Das et D. Lee. *Learning Spark – Lightning-Fast Data Analytics*, 2e édition. O'Reilly, 2020.
  - Chambers, B. et M. Zaharia. *Spark : The Definitive Guide*. O'Reilly, 2018.
  - Best Free Data Science eBooks (mise à jour 2020)
- Visualisation des données
  - B. Fry. *Visualizing Data*. O'Reilly, 2008.
  - R. Spence. *Information Visualization : Design for Interaction*. Prentice Hall, 2007.
  - T. Munzner. *Visualization Analysis and Design*. A K Peters / CRC Press, 2014.