Apprentissage Automatique I MATH60629

Calcul parallèle pour données massives
— Week #10

Aujourd’hui

A. Parallélisme pour l’apprentissage automatique

- Entrée en matière
- Les GPU
- Courte introduction à MapReduce/Hadoop & Spark

B. Résumé de la matière

À noter: nous avons surtout utilisé des concepts statistiques dans le cours. Aujourd’hui nous utiliserons des concepts informatiques.

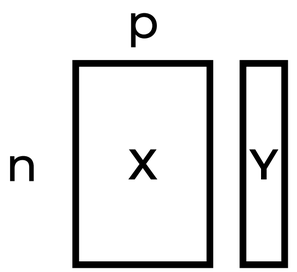
Données & Calculs

- On génère maintenant des quantités massives de données

  1. Google 4K recherches/s, Twitter: 6K tweets/s, Amazon: 100s produits vendus/s (source: internetlifestats.com)

  2. Banques, compagnies d'assurance, etc.

  3. Site web de taille moyenne



- n et p sont grands

- En général, le coût des calculs dépend de la taille des données

  - Souvent pour entraîner un modèle de ML, il faut une ou plusieurs opérations qui prennent en compte toutes les données

    p. ex., Régression linéaire $\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$

Difficultés liées aux données massives

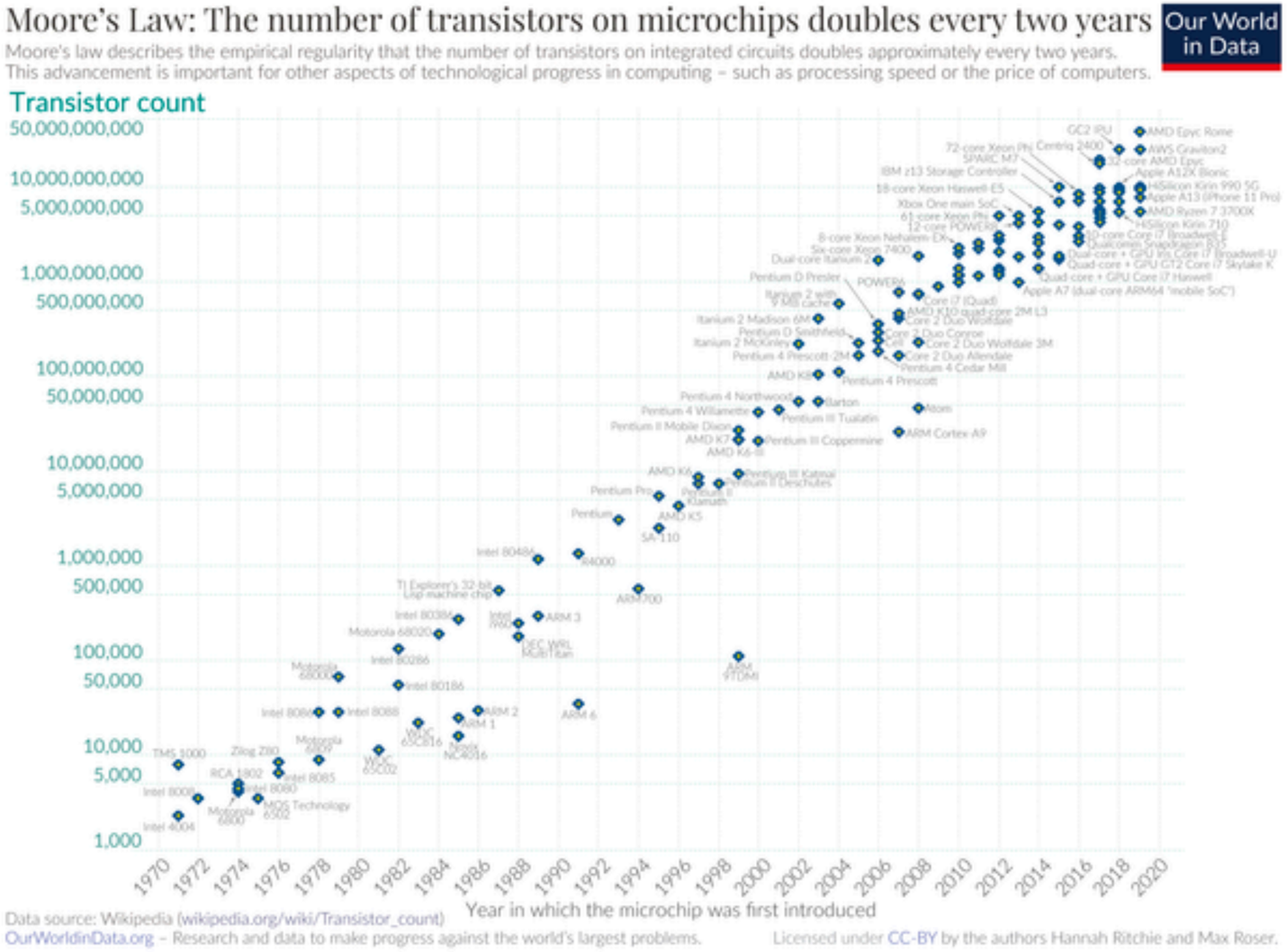
Bon marché

1. **Stockage**

Coût important

2. **Calcul (entraînement, test)**

Moore’s Law



[https://en.wikipedia.org/wiki/Moore%27s_law]

Comment augmenter la puissance de calculs

- Floating point operations per second (Flop)
- Téléphone ~ 11 TFlops
- 1 Tera: 1,000 Giga, 1 Peta: 1,000 Tera, 1 Exa: 1,000 Peta

1. “Un” ordinateur

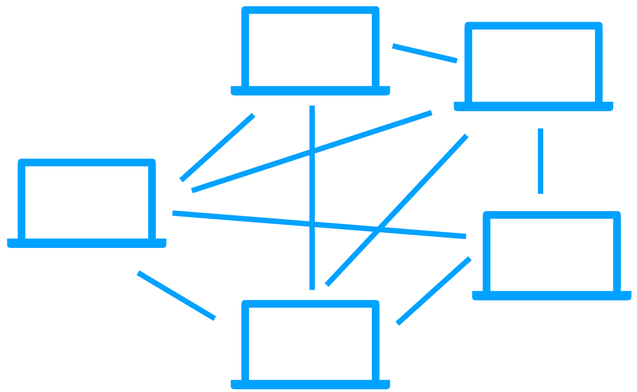
- Super Ordinateur
  - 1 ExaFlop*

https://www.top500.org/lists/top500/list/2020/06/

Photo from Riken

3. Calcul distribué

- 2,3 ExaFlops* (Folding@home)



2. Matériel spécialisé

- On optimise pour certaines d’opérations (calculs)
  - Graphical Processing Unit (GPU), Field Programmable Gated Array (FPGA)
  - ~1,000 TFlops

*: ces chiffres sont une indication et sont sujets à changement

Remarque: Les supers ordinateurs utilisent aussi le calcul distribué

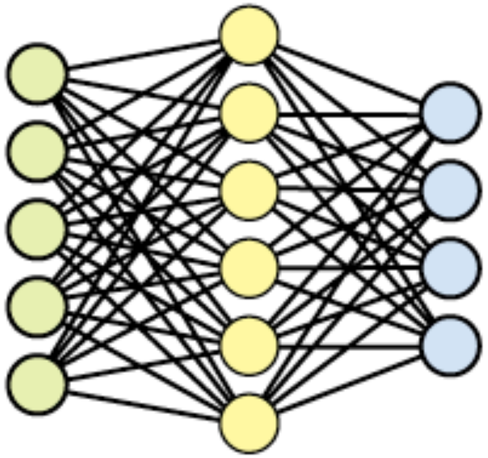
GPU et autres matériels spécialisés

Matériel

http://www.personal.psu.edu/users/d/l/dlm99/cpu.html

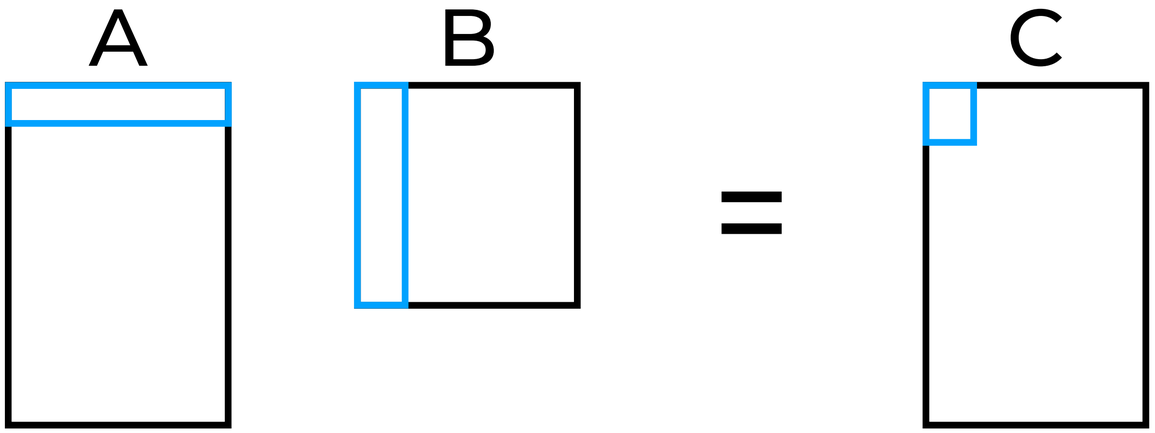
- Central Processing Unit (CPU)
  - Cerveau de l’ordinateur
  - Exécute toutes les instructions venant des programmes
    - Calculs arithmétiques, lectures/écritures, etc.

Réseau de neurones

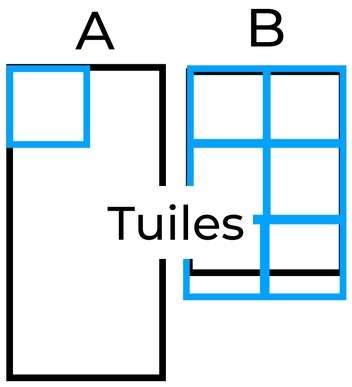


- Algèbre linéaire:
  - Multiplication *vecteur X matrice*
    - Calculer l’activation d’un neurone
    - Calculer l’activation d’un neurone pour plusieurs données
  - Multiplication matrice X matrice
    - Calculer l’activation de plusieurs neurones pour plusieurs données
- Les dimensions de ces objets dépendent de la taille des données (mini batch) et la taille des réseaux

Paralléliser un réseau de neurones



$C = AB$



- Les valeurs de C peuvent être calculées indépendamment les unes des autres
- Les multiplications de matrices peuvent être parallélisées
- En parallélisant on peut donc obtenir des gains computationnels très significatifs
  - Fonction de la taille de C
- En pratique on découpe en tuiles
- Note: ce n'est pas le cas pour toutes les opérations sur les matrices. Notamment calculer l'inverse d'une matrice.

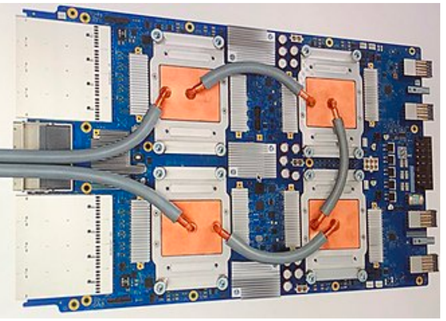
Matériel spécialisé

NVidia

- Graphical Processing Unit (GPU)
  - Initialement utilisé pour les jeux
  - Spécialisé pour des opérations d'algèbre linéaire
    - Milliers de coeurs à la fois (versus qq. dizaines pour les CPU)
    - Accès rapide à la mémoire
  - Plusieurs GPU par ordinateur

https://cryptomining-blog.com/tag/multi-gpu-mining-rig/

Matériel encore plus spécialisé



https://en.wikipedia.org/wiki/Tensor_Processing_Unit

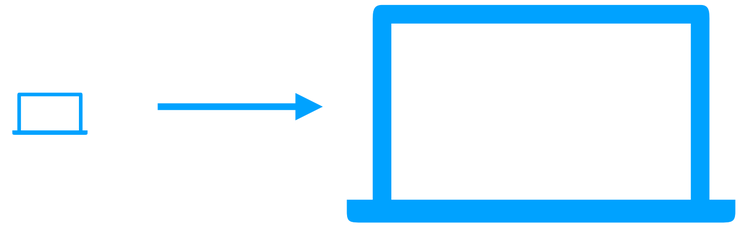
- Tensorflow Processing Unit (TPU)
  - Développé par Google pour les réseaux de neurones
  - Que des opérations de multiplications de matrices
  - Entraînement et test
  - Précision des opérations moins élevée que les GPU
  - Plusieurs TPU par “ordinateur”

https://colab.research.google.com/github/lcharlin/80-629/blob/master/week10-ParallelComputations/CPU_GPU_TPU.ipynb

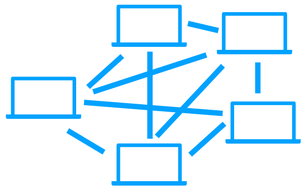
- Il faut peu de changements à notre code pour utiliser un GPU

  - Attention aux transferts CPU-GPU

Calcul distribué



- Des ordinateurs plus rapides peuvent aider
- Qu’en est-il d’ordinateurs travaillant ensemble?



  - Diviser le problème en plus petits problèmes

    1. Les ordinateurs calculent en parallèle la solution des petits problèmes

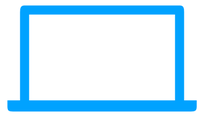
    2. On combine ensuite la solution de chaque petit problème en une solution à notre problème initial

Un exemple tout simple

- Vous devez compter le nombre de bâtiments à Montréal

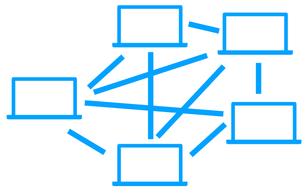
1. Centralisé (single computer):

   - Demander à un coureur de marathon de compter les maisons tout seul



2. Distribué (plusieurs ordinateurs):

   - Demander à 1 000 personnes de chacune compter les maisons dans une petite partie de la ville
   - Une fois terminés, ils donnent leur résultat au quartier général



Calculs distribués avec MapReduce

Deux approches

- MapReduce
  - Fundamentals and bag-of-words example
- Spark
  - Fundamentals & MLlib

MapReduce

- Créé pas des ingénieurs chez Google

  “MapReduce: Simplified Data Processing on Large Clusters”, Jeffrey Dean and Sanjay Ghemawat, 2004

  - Maintenant aussi connu sous le nom de (Apache) Hadoop
  - (Google a bâti sa capacité de calcul à partir d’ordinateurs “standards”)

- Interface pour le calcul distribué
  - Utile pour les algorithmes pouvant utiliser cette interface

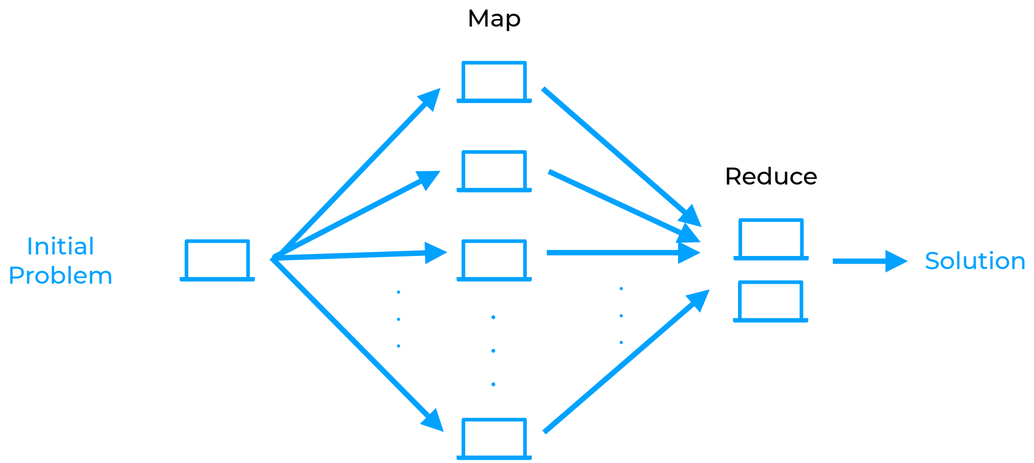
Interface MapReduce

- Deux types de tâches:

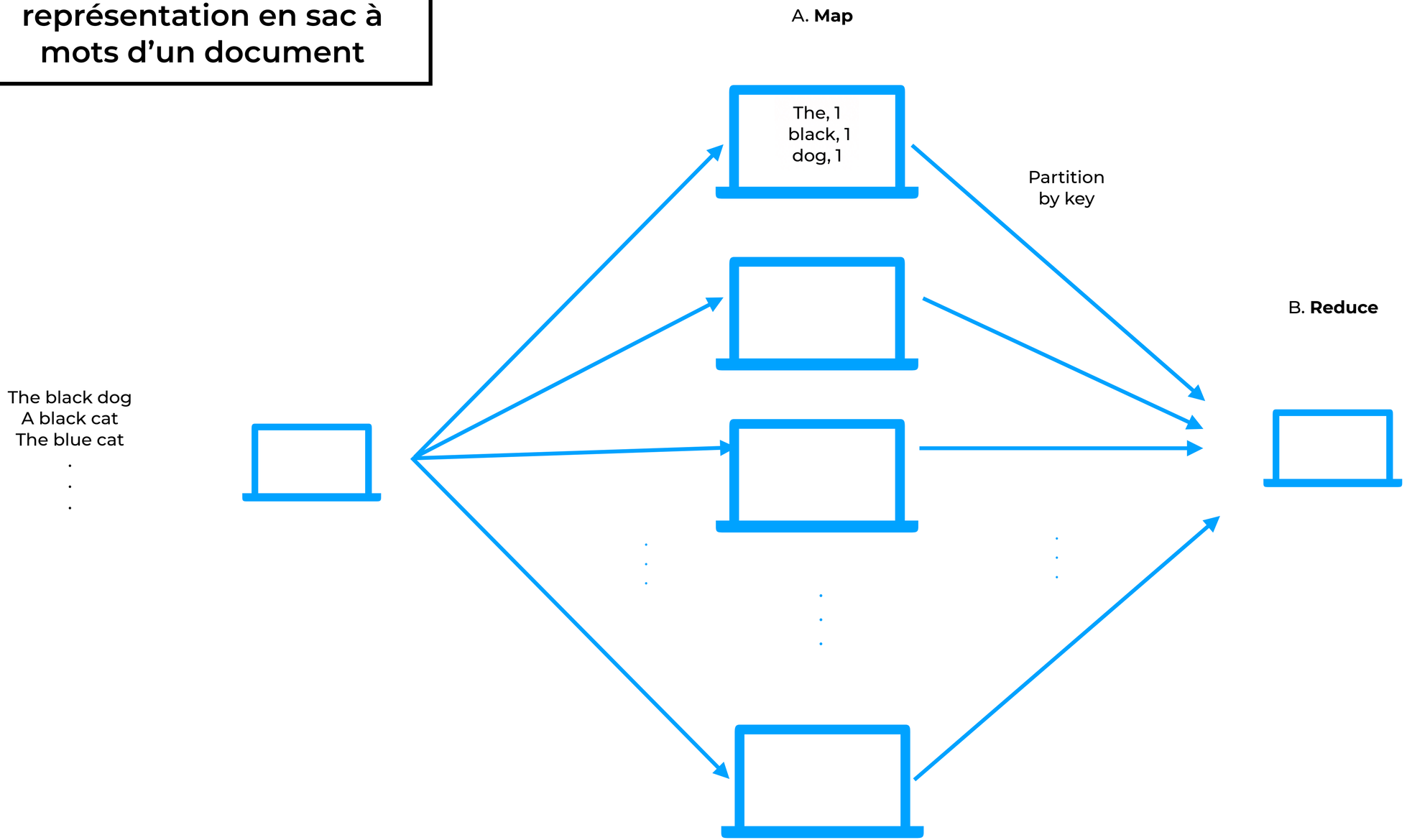
  A. Map: pour résoudre un sous-problème (op. de filtrage)

     - Ensuite une opération de tri est effectuée sur la sortie

  B. Reduce: combiner les résultats des « Mappers » (op. sommaire)



Tâche: Créez une représentation en sac à mots d'un document



Quelques détails

- Normalement, le nombre de sous-problèmes est plus grand que le nombre de machines
  - Donc le gain en temps de calcul est (au mieux) ~linéaire par rapport au nombre de machines
- Si un « mapper » ne résous pas sa tâche suffisamment rapidement on peut renvoyer la tâche à un autre
- Chargement et Écriture du données
  - Les données sont chargées du disque au début de chaque map et reduce
  - Les données sont écrites sur le disque à la fin de chaque map et reduce

MapReduce est très versatile

- Quand j'étais à Google « on » disait :

"Si votre problème ne peut pas utiliser l'interface MapReduce, c'est que vous n'y avez pas suffisamment réfléchi."

- Intuition: Votre problème doit être décomposable en des fonctions « map » et des fonction « reduce »

- Quelques exemples de problèmes "map-reduceable":

  - Ordonner, filtrer, trouver les valeurs uniques, statistiques standards

  - Aussi des tâches plus complexes: Requêtes SQL

MapReduce pour L'apprentissage automatique

1. Entraîner la régression linéaire
   - Rappel: il existe une solution analytique

$$w = (X^\top X)^{-1} X^\top Y$$

$$w = \left(\sum_{ij} X_i^\top X_j\right)^{-1} \left(\sum_{i} X_i^\top Y_i\right)$$

   - Tous les termes dans les sommations peuvent être calculés indépendamment

A. **Map**

$X_0^\top X_1$

2. D'autres modèles que nous avons étudiés ont aussi des solutions analytiques (p.ex., Naive Bayes)

3. Recherche d'hyperparamètres
   - Chaque mapper entraîne un modèle avec un ensemble d'hyperparamètres
   - Le reduce conserve l'erreur du meilleur modèle

Limites de MapReduce

- Souvent en apprentissage les modèles sont entraînés de manière itérative
  - Descente de gradient:
    1. Trouve les gradients de la fonction pour les paramètres actuels
    2. Mises à jour des paramètres avec le gradient
- Pas idéal pour le MapReduce
  - Demande plusieurs itérations de MapReduce
    - Les itérations sont séquentielles
  - Après chaque itération les données sont lues du disque et les résultats sont écrits sur le disque

Calcul distribué en utilisant Apache Spark

(Apache) Spark

- Avantages par rapport à MapReduce

1. Graphe computationnel moins restrictif

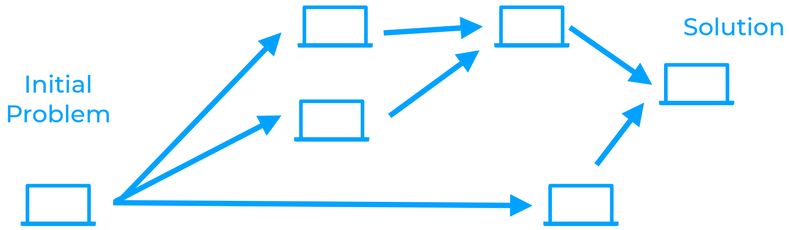
   (DAG à la place d'une opération Map suivie d'un Reduce)

   - Pas besoin d'écrire les résultats sur le disque après chaque ensemble d'opérations

2. Transformations plus riches

   - map, filtrer, produit cartésien, union, intersection, distincte, etc.

3. Tout reste en mémoire vive



Historique de Spark

- Création au laboratoire AMPLab de Berkeley (2009)
- Version 1.0 2014
  - Utilise les Resilient Distributed Datasets (RDDs)
- Version 2.0 juin 2016
- V2.4.4 septembre 2019, V3.0.1 septembre 2020, V3.5.4 September 2024
- Python + Spark: pySpark
- Pour commencer à utiliser Spark:
  1. “Advanced Analytics with Spark”, 2nd edition (2017).

     “Spark: The Definitive Guide: Big Data Processing Made Simple”, 2018
  2. Documentation du projet: https://spark.apache.org/docs/latest/

DataFrames

- An extra abstraction on top of RDDs
  - Encodes rows as a set of columns
    - Each column has a defined type
  - Useful for (pre-processed) machine learning datasets
- Same name as `data.frame` (R) or `pandas.DataFrame`
  - Similar type of abstraction but for distributed datasets
- Two types of operations (for our needs): transformers, estimators.

Descente de gradient parallèle

- Régression logistique

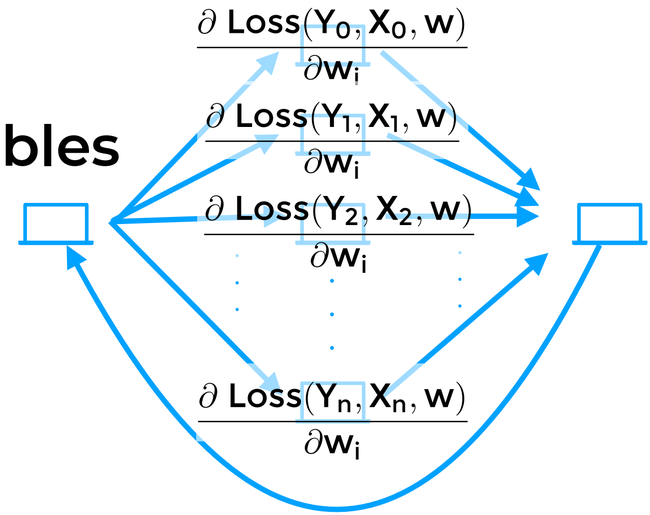
$$y = \frac{1}{1 + \exp(-w_0 - w_1 x_1 - w_2 x_2 - \ldots - w_p x_p)}$$

- Il n'existe pas de solution analytique, mais on peut utiliser la descente de gradient

$$\frac{\partial \, \text{Loss}(Y, X, w)}{\partial w_i}$$

- Les fonctions de perte sont souvent décomposables

$$\frac{\partial \, \sum_j \text{Loss}(Y_j, X_j, w)}{\partial w_i}$$



Spark's "Hello World"

DataFrame

```
data = spark.read.format("libsvm").load("hdfs://...")

model = LogisticRegression(regParam=0.01).fit(data)
```

Estimator

ML setup

0. Grappe Spark

Disponible en quelques clics en info nuagique

1. Accès aux données

Il existe maintenant une Interface pandas dans pySpark

Documentations pandas pySpark

2. Libraire d'apprentissage automatique

Machine Learning Library (MLlib) Guide

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Classification and Regression - RDD-based API

The `spark.mllib` package supports various methods for binary classification, multiclass classification, and regression analysis. The table below outlines the supported algorithms for each type of problem.

Problem Type	Supported Methods
Binary Classification	linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes
Multiclass Classification	logistic regression, decision trees, random forests, naive Bayes
Regression	linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression

https://spark.apache.org/docs/latest/ml-guide.html

Points importants

- Le calcul parallèle est maintenant essentiel
  - Les GPU, TPU sont utilisés de facto pour entrainer les réseaux de neurones
- Le calcul distribué est aussi très utile surtout si:
  - Les données sont immenses
  - Pour obtenir des gains encore plus importants
- Les libraires actuelles (p.ex., spark) deviennent petit à petit aussi faciles à utiliser que sklearn
  - On obtient des gains computationnels en décomposant les problèmes en sous-problèmes
  - Encore besoin d'ingénierie