Introduction aux algorithmes MapReduce

Mathieu Dumoulin (GRAAL), 14 Février 2014

Plan

- Introduction de la problématique
- Tutoriel MapReduce
- Design d'algorithmes MapReduce
  - Tri, somme et calcul de moyenne
- PageRank en MapReduce
- Conclusion

« Big Data »

*Big data is **high volume**, **high velocity**, and/or **high variety** information assets that **require new forms of processing** to enable enhanced decision making, insight discovery and process optimization*

- Gartner, updated definition of big data (2012)

*Building **new analytic** applications based on **new types of data**, in order to better serve your customers and **drive a better competitive advantage***

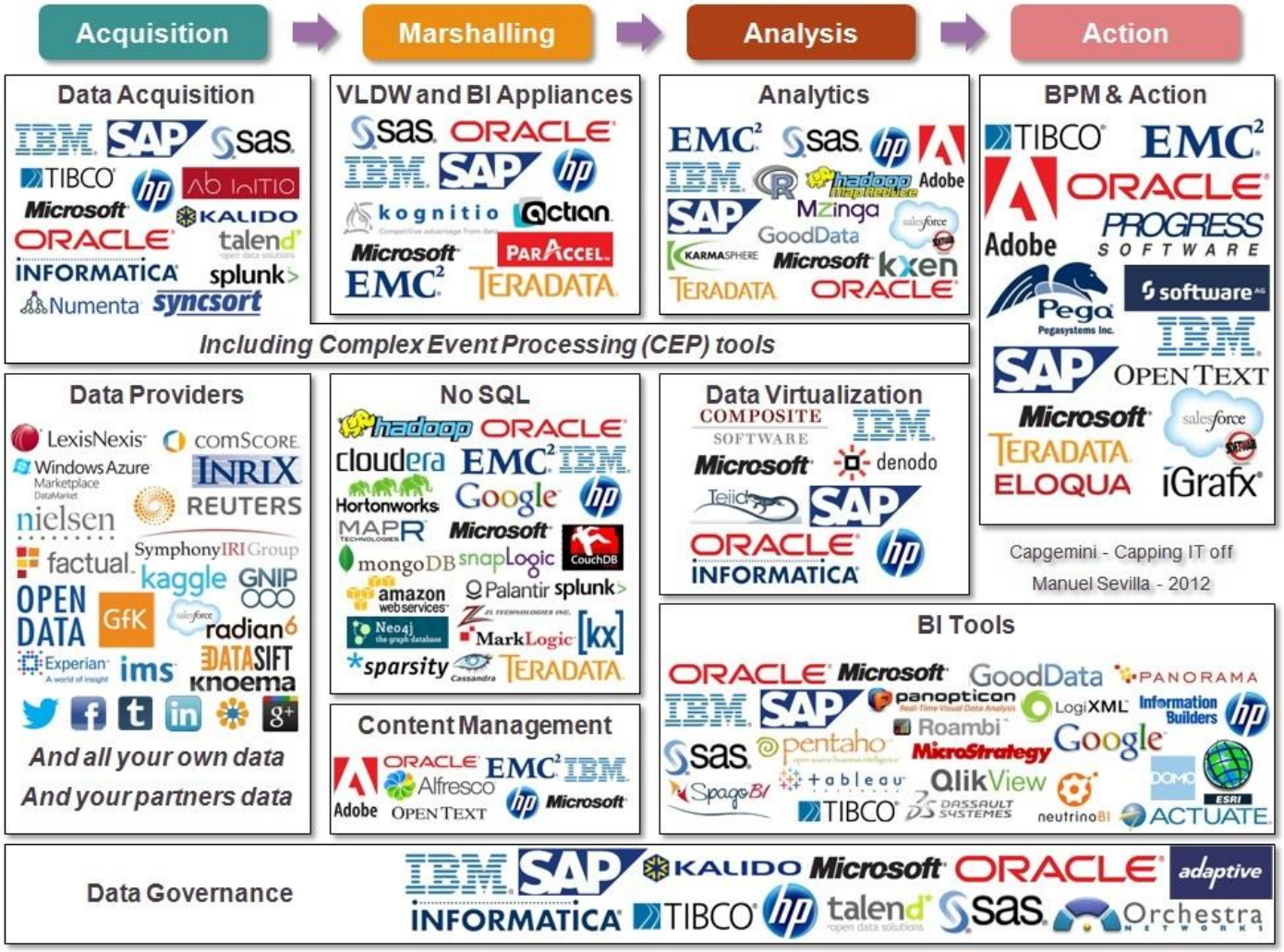
- David McJannet, Hortonworks

Un outil spécialisé

Problèmes où Hadoop est envisageable:

- Trop de données (GB,TB,PB)
- Améliorer des résultats existants
- Obtenir de nouveaux résultats
- Combiner des données hétérogènes
- Croissance rapide (et constante) des données
- Temps de traitement lent (minutes, heures)
- Budgets limités
- Plusieurs ordinateurs déjà disponibles

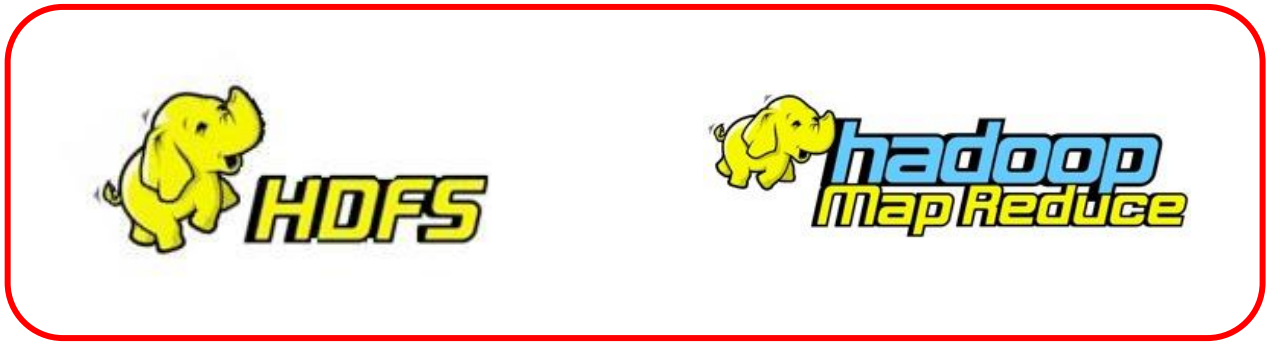
Hadoop au cœur du big data

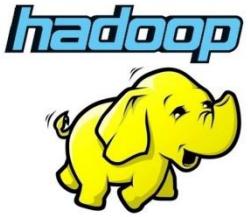


MapReduce au cœur de Hadoop

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

- Hadoop Tutorial, hadoop.apache.org





Gain d'utiliser Hadoop?

- Tout le travail de distribution, balancement de charge, synchronisation et de gestion d'erreur est géré automatiquement
- Il suffit de programmer `Map` et `Reduce` (Java, C++, Python, bash, etc.)
- Une grappe Hadoop peut évoluer facilement en ajoutant des nœuds en tout temps
- Hadoop offre un rapport performance-prix très compétitif (Amazon EMS, réutilisation de PC existants, aucun coûts de licences ni de matériel spécialisé HPC)

L'exemple WordCount

On veut trouver les k mots les plus fréquents dans une collection de textes.

```
def word_count(text, k):
    counts = defaultdict(int)
    for word in text.split():
        counts[word.lower()] += 1
return sorted(counts, key=counts.get, reverse=True)[:k]
```

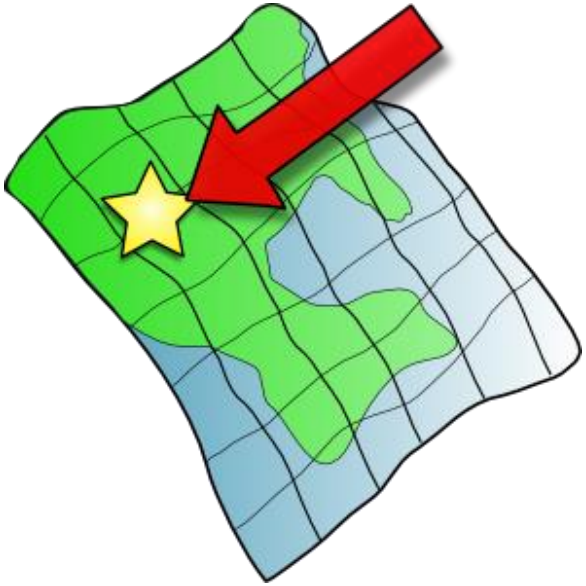
Mais cette solution est-elle la bonne si le texte est très grand?

Et s'il est très, très, …, très grand?

La taille en soit peut être un problème

Taille	Problème	Solution
• 100M mots	• Pas de problèmes	• Naïve avec 1 seul ordinateur
• 1000M mots	• Mémoire insuffisante	• Utiliser le disque, Fenêtre glissante
• 100MM mots	• Processeur insuffisant	• Multithreading, éliminer < N
• 1000MM mots	• 1 ordinateur insuffisant	• Distribuer le calcul
• Plus encore!	• Réseau insuffisant, contrôleur surchargé	• MapReduce (ou solution du même ordre comme MPI, etc.)

Tutoriel MapReduce



Map

Reduce

Map et Reduce: la paire Clef-Valeur

Mapper:

(K,V) → (K', V')

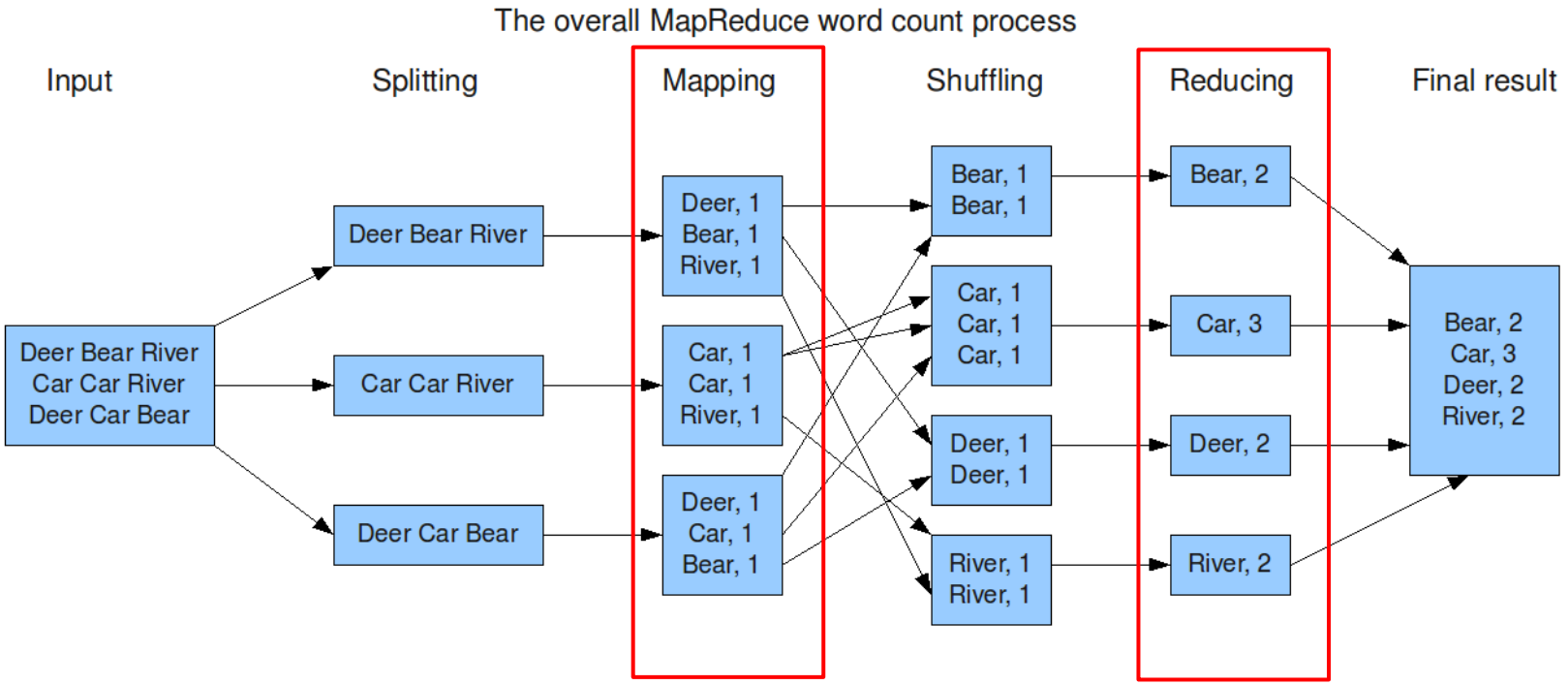
Données
(HDFS)

Reducer:

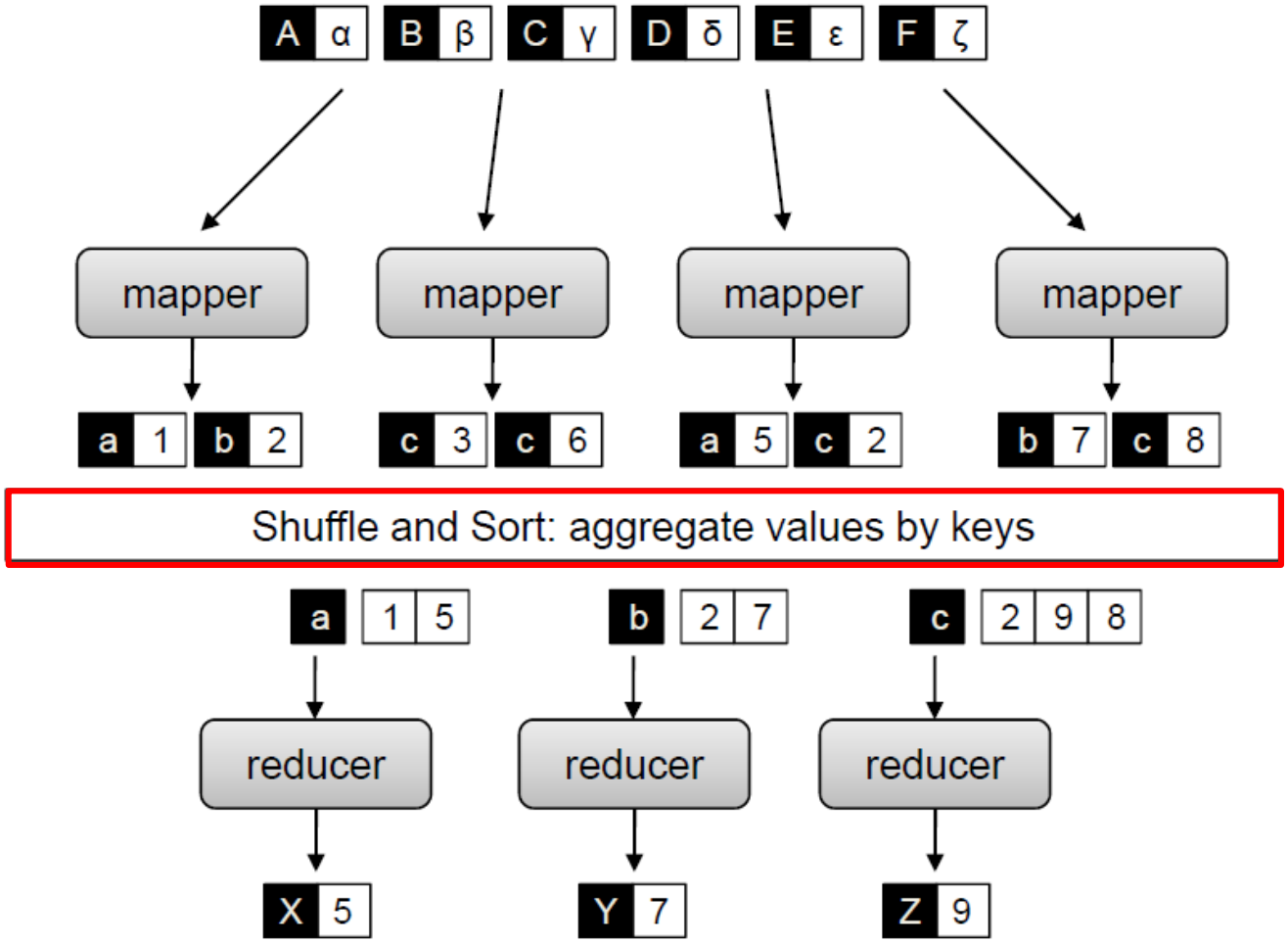
(K', [V', V',...]) → (K'', V'')

Données'
(HDFS)

MapReduce en action: WordCount illustré



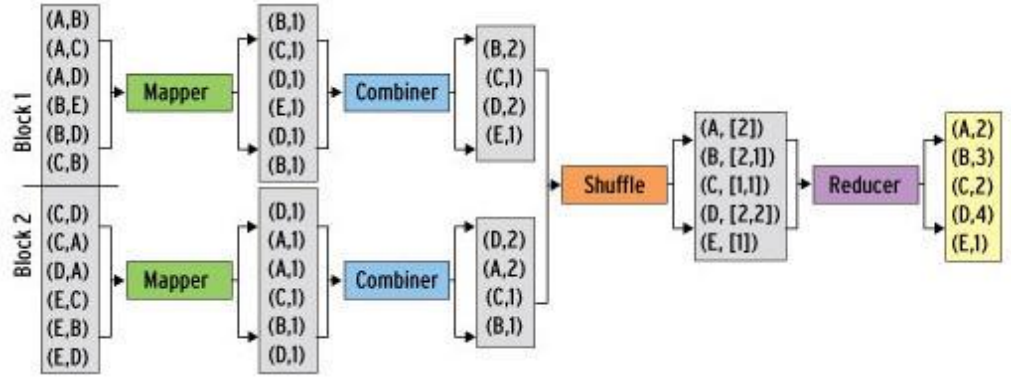
Map et Reduce: *Shuffle and Sort*

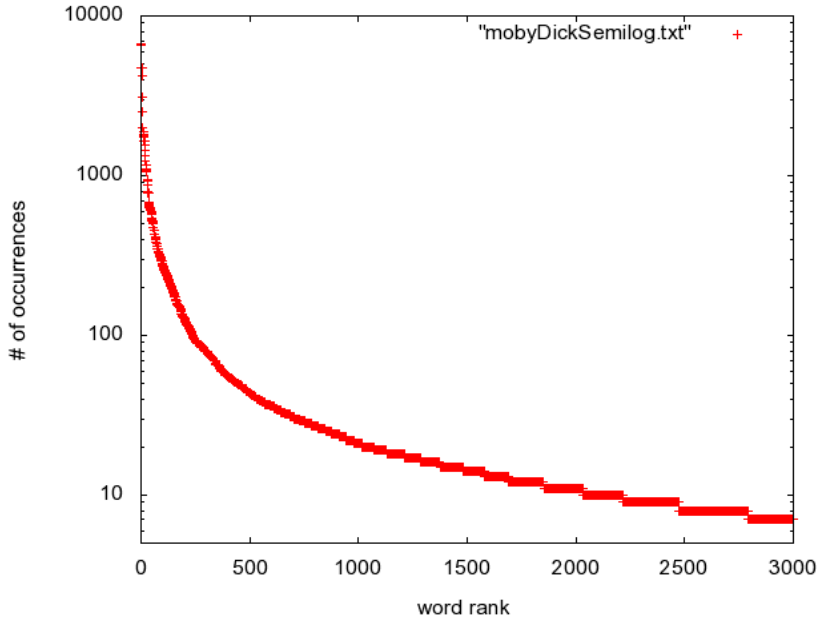


Source: Data Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, 2010

Map et Reduce (moins simplifié)

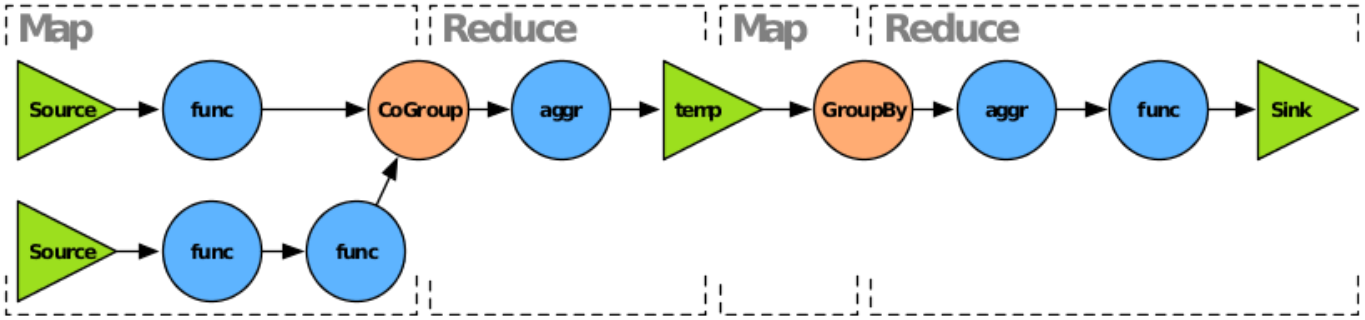
- Les vrai opérateurs:
  - Mapper
  - **Combiner**
  - **Partitioner**
  - Reducer



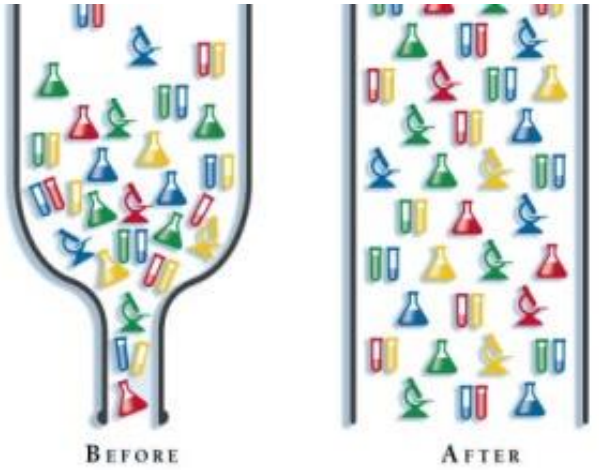


Design d'algorithmes pour MapReduce

1- Il faut reformuler les algorithmes en fonctionnel:



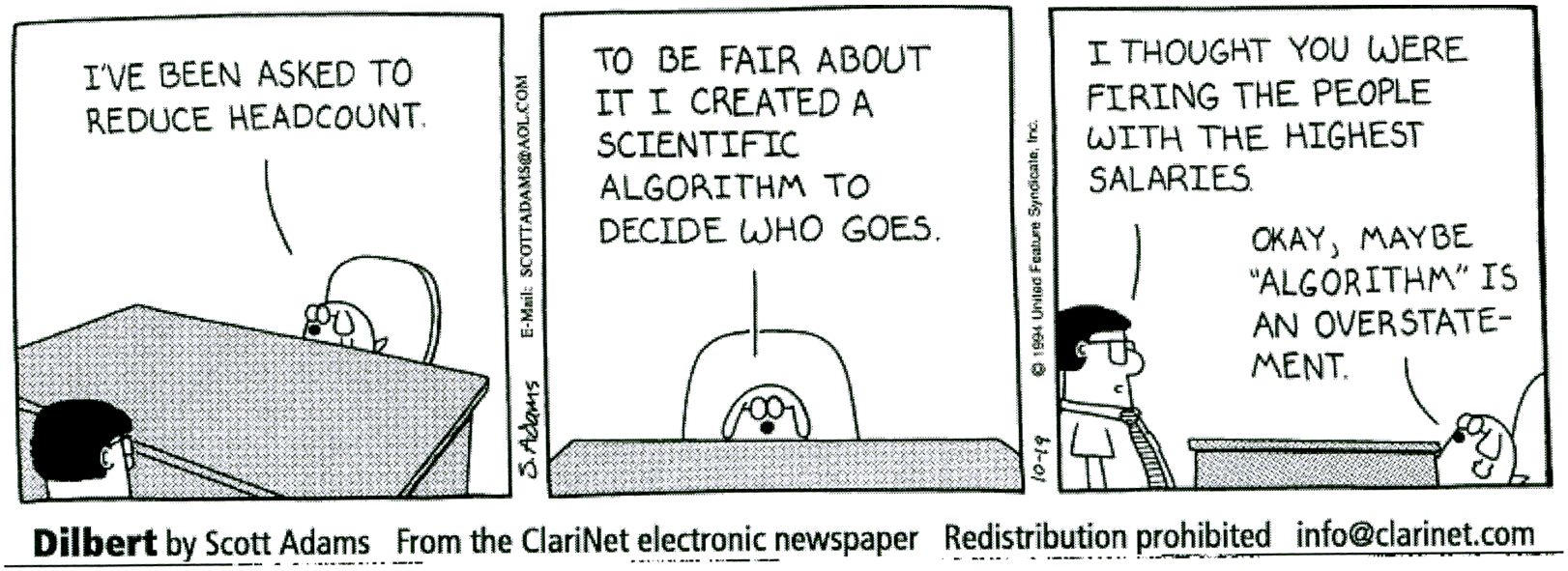
2- La bande passante du réseau est une ressource limitée à optimiser:



Exemples choisis

- Exemples simples:
  - Tri
  - Somme
  - Moyenne
- Un exemple complet: PageRank
- Bonus:

  K-Means (Lloyd's algorithm)



Trier en MapReduce

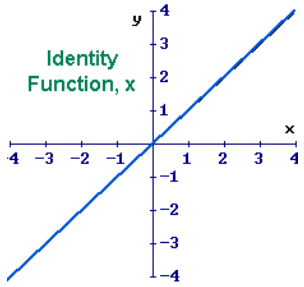
- Propriété du réducteur: les paires sont traitée dans l'ordre (selon la clef), les réducteurs sont aussi dans l'ordre

Mapper:

Émettre l'élément à trier comme nouvelle clef

Reducer:

Aucun (i.e. fonction identité)



Calcul d'une somme (WordCount)

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

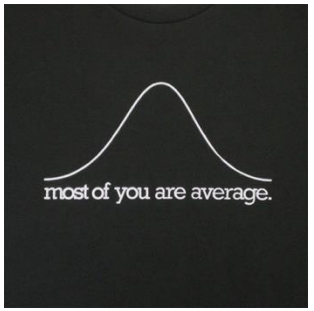
Amélioration: le *combiner*

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

Combiner = Reducer!

Calcul de moyenne




```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

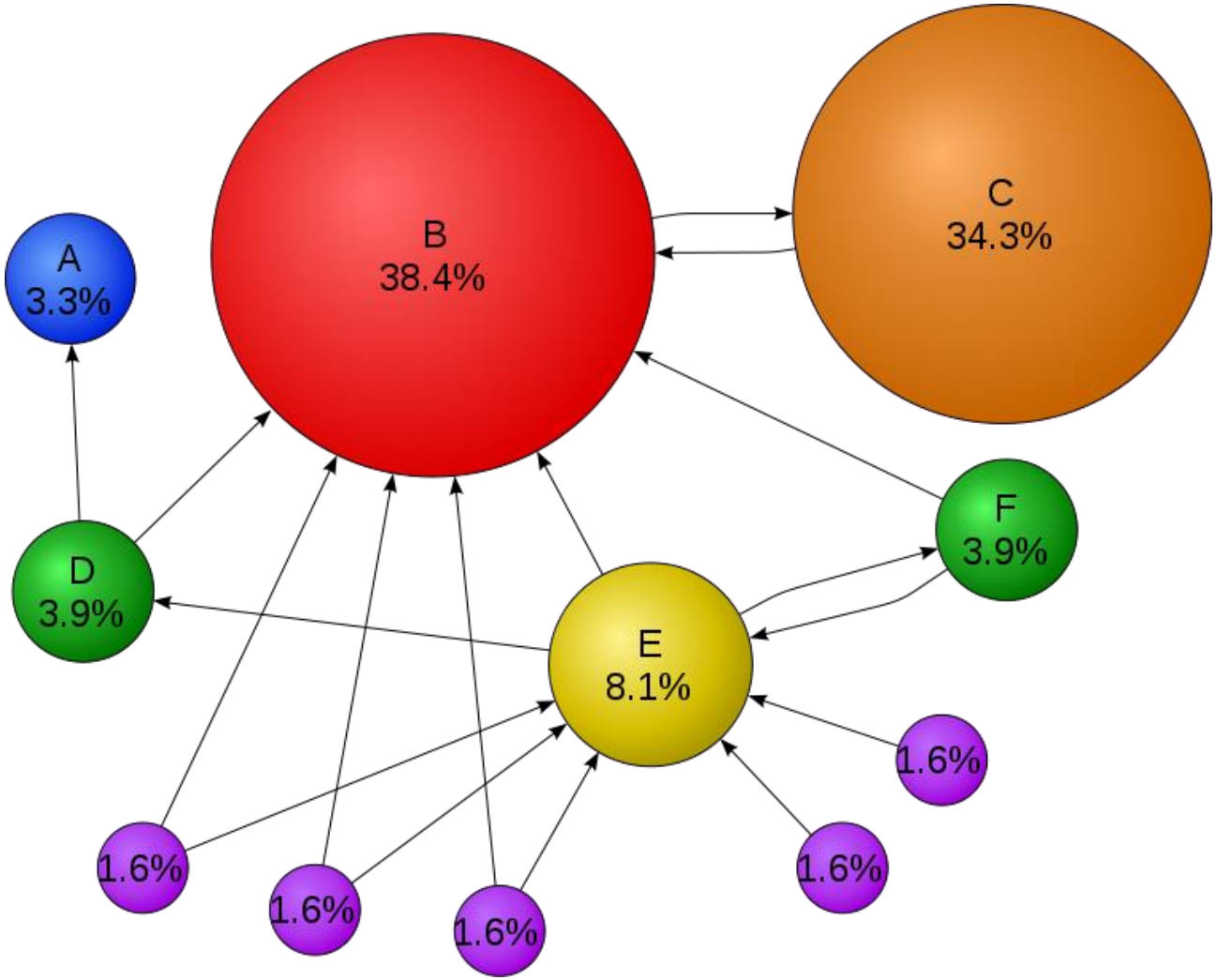
Utiliser un combiner est il approprié?

Si on reprend le reducer comme combiner, l'algorithme est-il encore correct?

Comment améliorer ce calcul?

```
1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))
```

Design d'une solution MapReduce pour l'algorithme PageRank

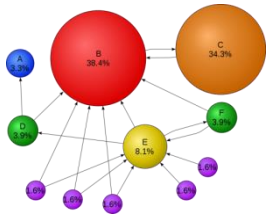


Qu'est-ce que PageRank?

Distribution de **probabilité** sur les pages web qui représente la chance qu'un utilisateur naviguant **au hasard** arrive à une page web particulière.

Notes:

- Le web est un graphe orienté, une page est un nœud et les hyperliens sont des arcs.
- L'algorithme recalcule la probabilité de toutes les pages **itérativement** jusqu'à convergence



Comment calculer PageRank (simplifié)

$$PR(p_i) = \sum_{p_i \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

- $p_1, p_2, \ldots, p_N$ sont les pages web (les nœuds du graphe)
- $M(p_i)$ est l'ensemble des pages ayant un lien vers p_i
- $L(p_j)$ est le nombre de liens sortant de la page p_j
- N est le nombre total de pages web

Note: Pour simplifier, on élimine le facteur d'atténuation, paramétrisé par la probabilité que l'utilisateur arrête de naviguer.

Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999) *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report. Stanford InfoLab.

PageRank par un exemple

Le web a trois pages web: A, B et C

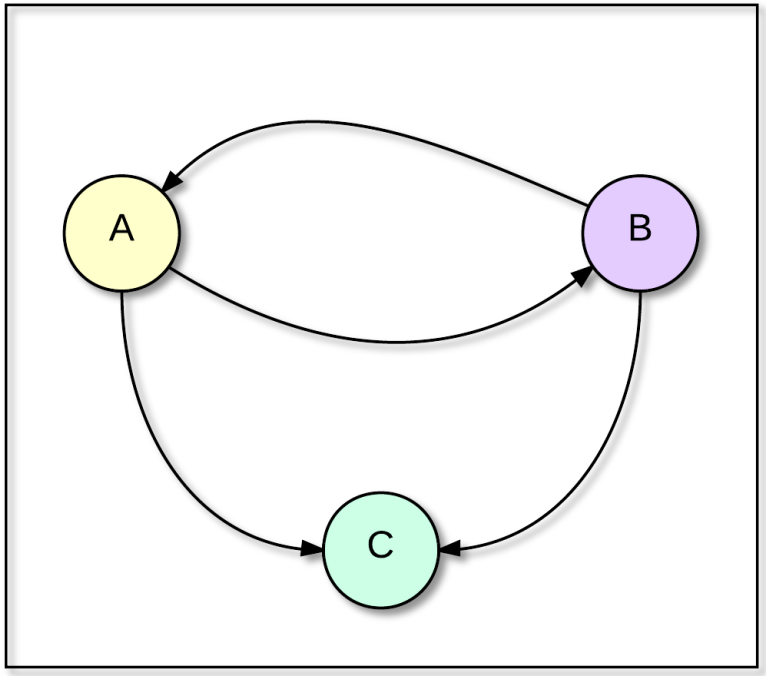
Initialisation: $PR(A) = PR(B) = PR(C) = 0.33$

Jusqu'à convergence:

$$PR(A) = \frac{PR(B)}{2}$$

$$PR(B) = \frac{PR(A)}{2}$$

$$PR(C) = \frac{PR(A)}{2} + \frac{PR(B)}{2}$$



PageRank en MapReduce

Donnés de départ:

collection de pages web (URL, [URL_{lien}])

1. Bâtir et initialiser le graphe
2. Jusqu'à convergence, recalculer PageRank pour chaque page web
3. Retourner les *K* premières valeurs de PageRank (pas présenté)

Étape 1: Bâtir le graphe

Mapper:

Entrée: une page web

Pour chaque lien de la page, émettre:

clef: URL_{page}

valeur: URL_{lien}

Reducer:

Entrée:

clef: URL_{page}

valeurs: $[URL_{lien}, \ldots]$

Sortie:

clef: URL_{page}

valeur: «PR; $[URL_{lien}]$»

Étape 2: calculer PageRank - Map

Mapper:

Entrée:

clef: URL_{page}

valeur: «PR; [$URL_{lien,\ldots}$]»

Sortie:

Pour chaque URL_{lien}, émettre:

clef: URL_{lien}

valeur: «URL_{page}; PR, nb_url_{lien}»

Où: nb_url_{lien} est le compte de URL_{lien}

Étape 2: calculer PageRank - Reduce

Reducer:

Entrée:

clef: URL_{page}

valeurs: [«$URL_{inverse}$;PR, $nb\_url_{page\_inverse}$», …]

Traitement: calculer le PR

Sortie:

clef: URL_{page}

valeurs: « PR;[URL_{lien}]»

PageRank en MapReduce: Résultats

Data set	No. of nodes	No. of edges	Avg No. of edges/node	Size
Cornell	626422	4477835	7	126 MB
edu	4527014	39874684	9	1.02 GB
Amazon	122047146	1378360637	11	45 GB

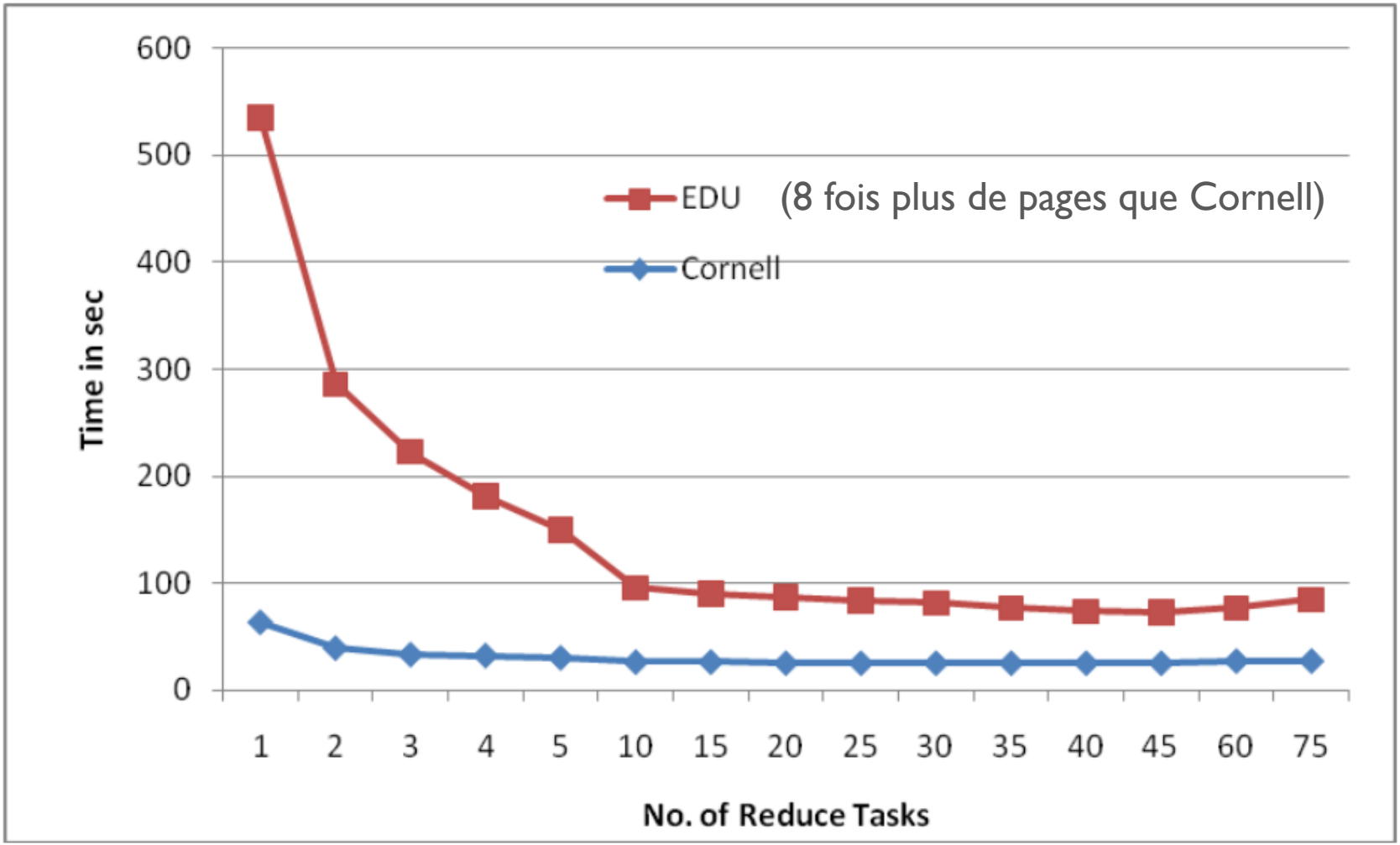
Table 1 : Details of the datasets used to test the page rank algorithm.

Data Set	Formatter	PageRank (50 iterations)	GetPageRank
Cornell	7 sec	22 min 10 sec	26 sec
edu	20 sec	47 min 44 sec	30 sec
Amazon	9 min 8 sec		

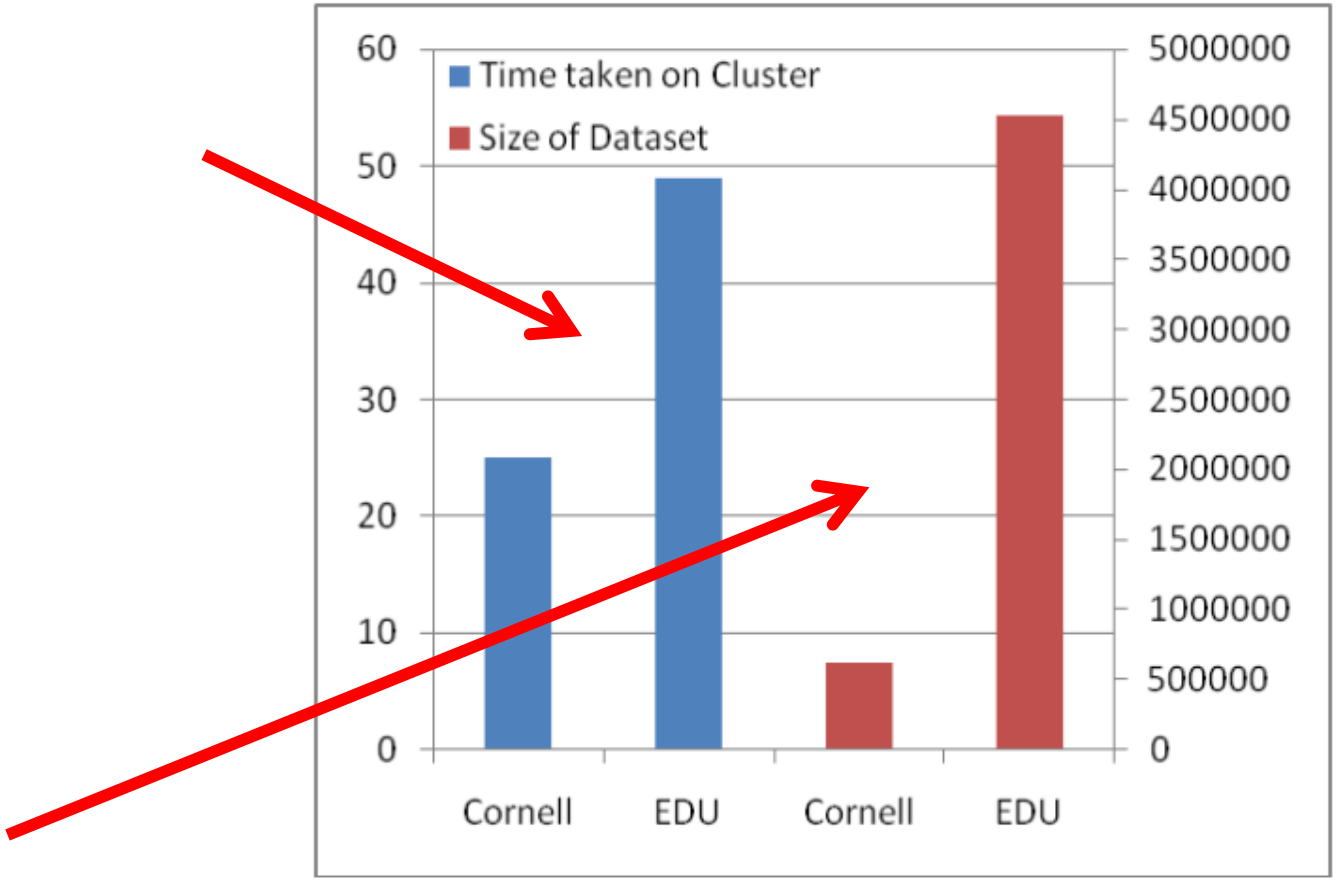
Table 2: Execution time taken by each of the three modules for 3 datasets.

Notes:
Source: **PageRank Calculation using Map Reduce** - The Cornell Web Lab (2008)
Résultats obtenus sur une grappe Hadoop de 50 nœuds (Intel Xeon 2.66GHz 16GB ram)
Mon implémentation: https://bitbucket.org/mathieu_dumoulin/pagerank-mr

MapReduce PageRank: Résultats



MapReduce PageRank: Résultats

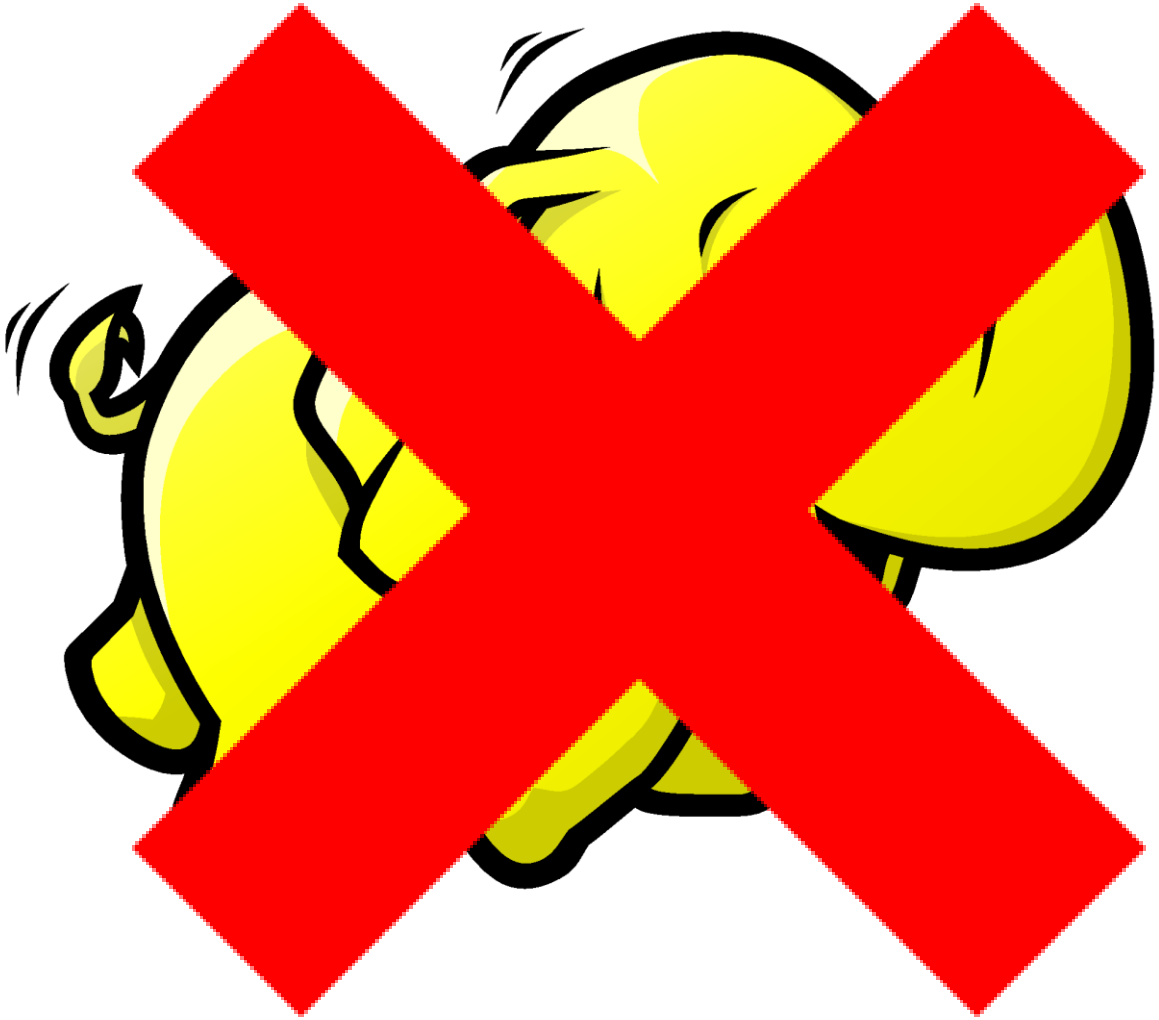


Graph 4: Effect of data size on MapReduce

Conclusion (malheureuse)

- MapReduce est une solution puissante au problème de programmation distribuée
- La plate-forme fait toute la distribution et la gestion des erreurs
- Le travail de programmation commence par la formulation d'un algorithme MapReduce
  - Ce n'est pas toujours facile
  - Trouver et formuler un algorithme performant est relativement difficile
  - Le travail réel de programmation demande une maîtrise (très) avancée de Java

Conclusion



Conclusion

On n'est pas prisonnier de MapReduce pour utiliser une grappe Hadoop!

Apache Pig: optimiser les tâches de ETL

Apache Hive: analyser ses données façon SQL

Cascading et Apache Crunch: librairies Java qui simplifient les opérations difficiles en MapReduce

Apache Mahout: librarie de machine learning qui peut utiliser une grappe Hadoop « automatiquement »

Questions et commentaires

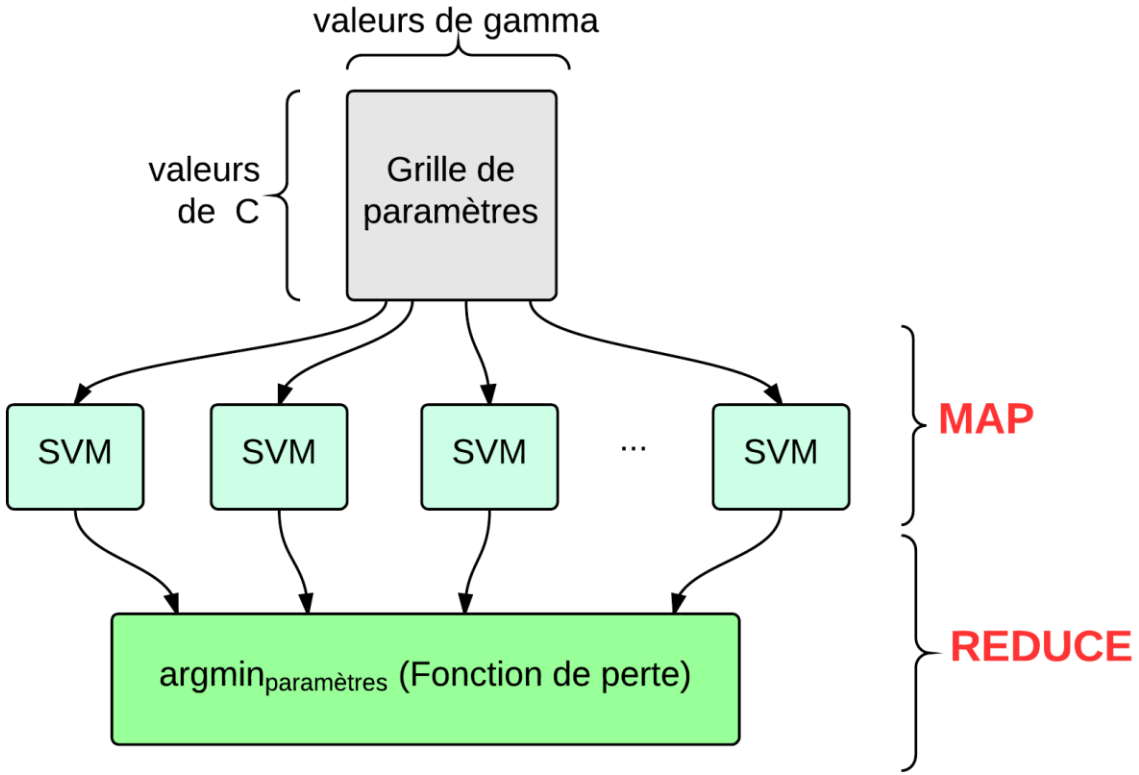
Hadoop est de plus en plus une composante d'infrastructure « standard » pour le traitement de donnée à grande échelle et est promis à un bel avenir!

Partie Bonus

- MapReduce et Machine Learning
  - Exemple
  - Algorithme K-Means

Map et Reduce et le machine learning?

Problématique exemple: recherche de paramètres optimaux



Yasser Ganjisaffar, Thomas Debeauvais, Sara Javanmardi, Rich Caruana, and Cristina Videira Lopes. 2011. Distributed tuning of machine learning algorithms using MapReduce Clusters. In *Proceedings of the Third Workshop on Large Scale Data Mining: Theory and Applications* (LDMTA '11). ACM, New York, NY, USA, , Article 2 , 8 pages. DOI=10.1145/2002945.2002947 http://doi.acm.org/10.1145/2002945.2002947

Bonus:
Algorithme *K-means clustering*

- Problème: regrouper des données en *K* groupes

How many clusters?

Six Clusters

Two Clusters

Four Clusters

Algorithme K-means (*Lloyd's Algorithm)*

Initialiser les K centroïdes (d'une certaine façon)

Pour chacune de plusieurs d'itérations:

- Pour chaque point:
  - Assigner au centroïde le plus proche
    - Selon une mesure de distance (ex: distance Euclidienne)
- Pour chaque centroïde:
  - Recalculer sa position en faisant la moyenne des membres de son groupe

K-means en MapReduce

- Driver: lancer les itérations
- Combien d'étapes map-reduce?

Une seule!

K-means MapReduce

- Driver: le "main" qui lance les tâches (Job) MapReduce et qui itère jusqu'à convergence
- Mapper: Assigner chaque point au cluster le plus proche (en parallèle!)
  - Entrée: Key: Null value: vecteur
  - Sortie: Key: index du centroïde le plus proche, value: vecteur
- Reducer: Calculer la moyenne des points membre du cluster (en parallèle!)
  - Key:
- Information partagée: les vecteurs des centroïdes

K-Means et MapReduce: État de l'art

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Scalable k-means++. *Proc. VLDB Endow.* 5, 7 (March 2012), 622-633.

http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf

Implémenté dans la librairie Apache Mahout