III/Masses de données (Big Data) : GFS (Hadoop)

Objectifs

- Système de fichiers extensible, *passant à l'échelle*
- Applications manipulant de *gros volumes* de données
- Hautes performances pour un grand nombre de clients
- *Tolérant aux fautes* sur du matériel standard, non spécialisé

Exemple d'applications

- stockage de vidéo (youtube),
- moteur de recherche (google),
- transactions commerciales passées (amazon),

Contexte d'utilisation

- Les hypothèses habituelles (fichiers majoritairement petits, courte durée de vie, peu de modifications concurrentes) non sont plus valides :
  - ◊ Fichiers volumineux : plusieurs TB est très fréquent
  - ◊ Nombre de fichiers faible par rapport au volume (quelques millions)
  - ◊ Accès massifs à des données partagées rémanentes
- Pannes fréquentes : milliers de serveurs de stockage, matériel standard

Principe de conception

Privilégier la force brute

Accès aux fichiers

- Lecture majoritairement séquentielle, rarement directe
- Ecriture majoritairement en mode ajout, rarement directe
- Deux sous-classes :
  - ◊ Fichiers créés puis surtout lus (ex : vidéos)
  - ◊ Fichiers constamment en ajout par des centaines d'applications, peu lus (ex : log)
- Applications dédiées → cohérence relâchée

Interface

Désignation des fichiers

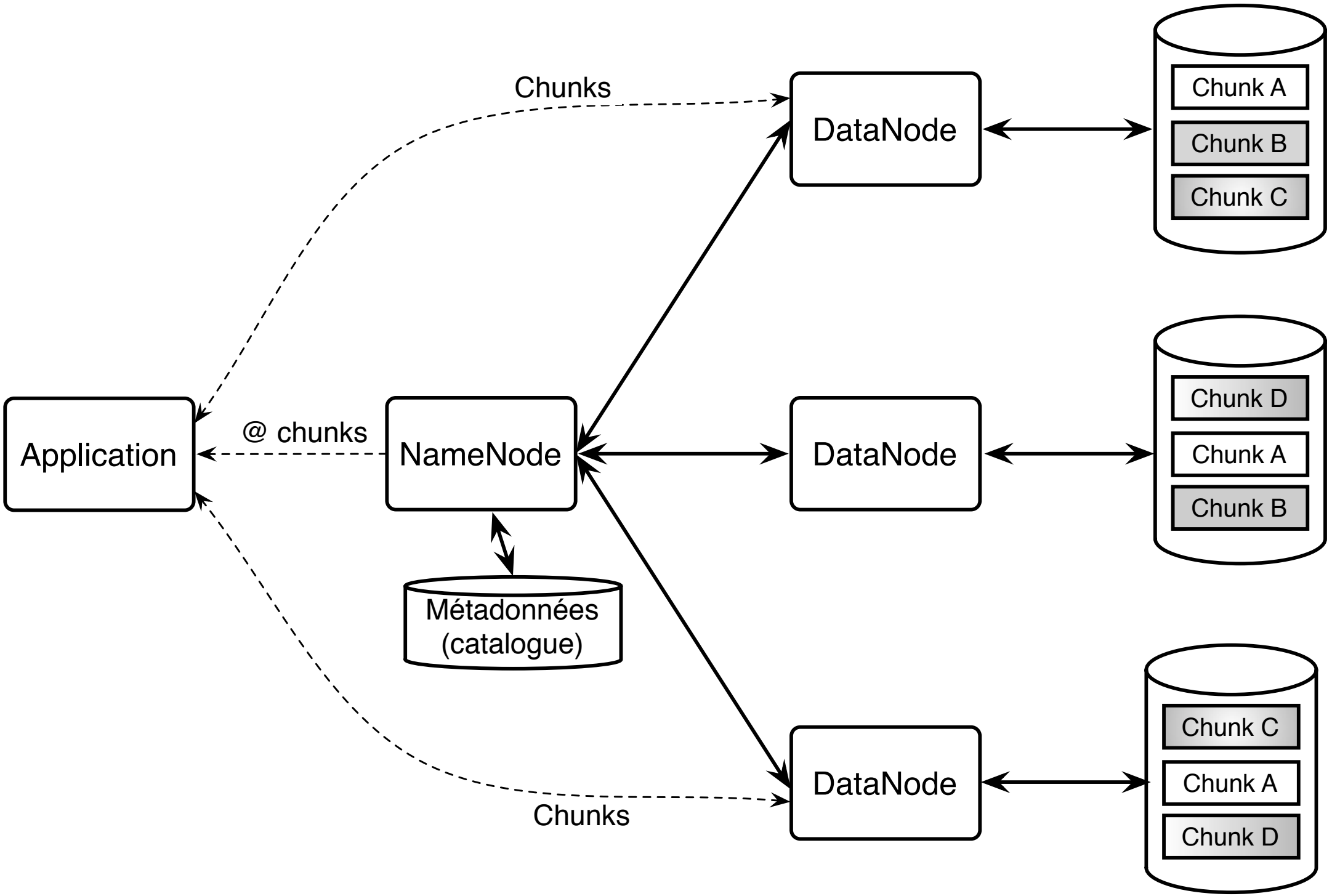
Espace de nommage plat (non hiérarchique)

Opérations

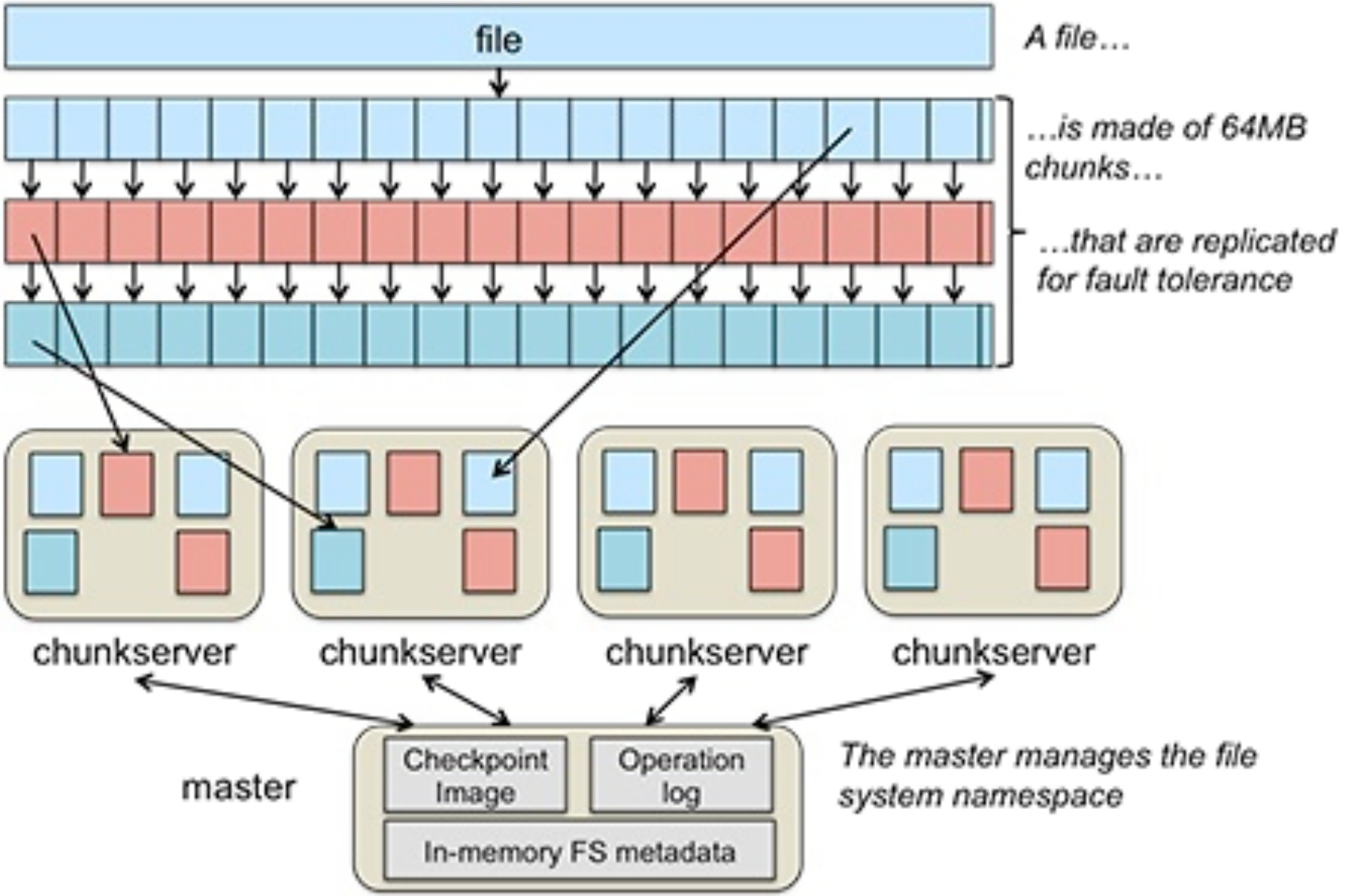
Usuelles : *create, delete, open, close, read, write*

Spécifiques :

- *append* : partage en écriture, sans verrouillage
- *snapshot* : copie efficace d'un fichier ou d'une arborescence, sans verrouillage


Chunks
DataNode
Chunk A
Chunk B
Chunk C
Application
@ chunks
NameNode
DataNode
Chunk D
Chunk A
Chunk B
Métadonnées
(catalogue)
DataNode
Chunk C
Chunk A
Chunk D
Chunks

Mise en œuvre de l'accès aux fichiers (source : Paul Krzyzanowski)



Serveur maître

- Métadonnées des fichiers
  - ◊ *Désignation* : simple table de hachage (fichier applicatif contenant une liste de noms)
    Pas de gestion de répertoires.
  - ◊ *Contrôle d'accès*
  - ◊ *Table d'implantation des fichiers* → emplacement des chunks
  - ◊ métadonnées en mémoire (relativement petites) pour performance
    → pas de cache côté serveur
- Maintenance
  - ◊ Verrouillage de chunks
  - ◊ Allocation/désallocation de chunks, ramasse-miettes
  - ◊ Migration de chunk (équilibrage, arrêt de machines)
- Répliqué (schéma maître-esclave) pour tolérance aux fautes

Ecriture d'un fichier

1 - *requête d'écriture* auprès du maître
→ un chunk responsable (primaire) + un nouveau numéro de version

2 - *envoi des données*

- ◊ le client envoie ses données à écrire au serveur de chunk primaire
- ◊ le primaire propage à l'une des copies (secondaire), qui propage à la suivante, etc…
- ◊ les données sont conservées en mémoire, non écrites

3 - *écriture des données*

- ◊ le client attend l'acquittement de toutes les copies (ou de la dernière)
- ◊ il envoie un message de validation au primaire
- ◊ le primaire ordonne les écritures (numéro de version), effectue son écriture et informe les copies secondaires
- ◊ les secondaires effectuent l'écriture et l'acquittent au primaire.
- ◊ l'ordre des opérations est identique pour tous (numéro de version)

4 - le primaire *informe le client*

Hadoop

Plateforme

- pour le traitement de masses de données selon le patron de conception *map/reduce*
- basé sur un système de fichiers (*HDFS*) dérivé de GFS

Schéma map/reduce

Idée

paralléliser un traitement élémentaire sur un grand ensemble de données de même structure en fragmentant cet ensemble et en traitant simultanément chaque fragment.

Réalisation

1 - *Map* (affectation des traitements) : on décompose l'ensemble de données en fragments, et on alloue un ouvrier à chaque fragment (l'allocation peut être dynamique)
Chaque ouvrier traite (typiquement filtrage + tri) son fragment en parallèle avec les autres

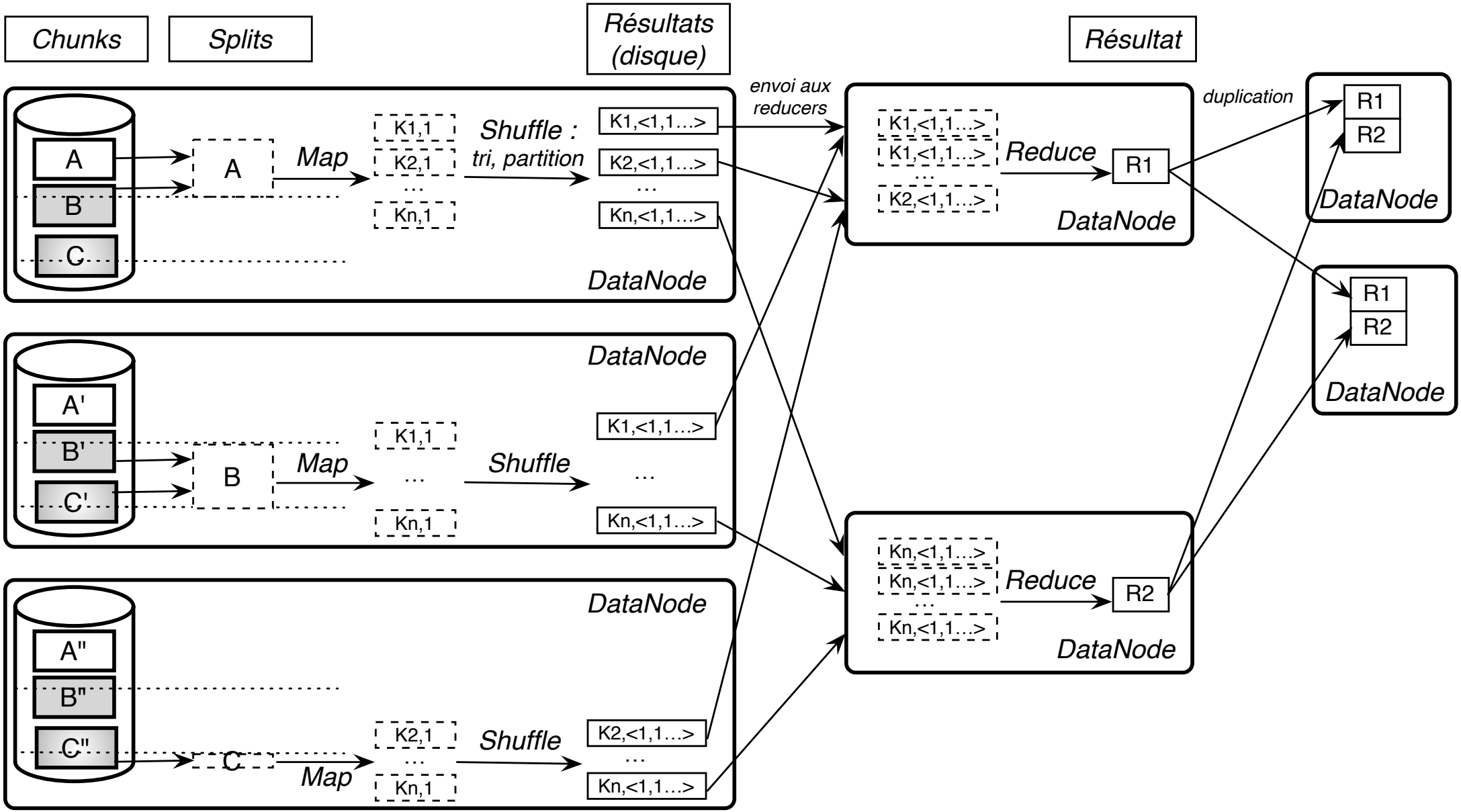
2 - *Reduce* (fusion) : lorsque les ouvriers ont terminé, leurs résultats sont agrégés (possibilité de redistribuer les résultats partiels (shuffle) pour une fusion parallèle/hiérarchique)

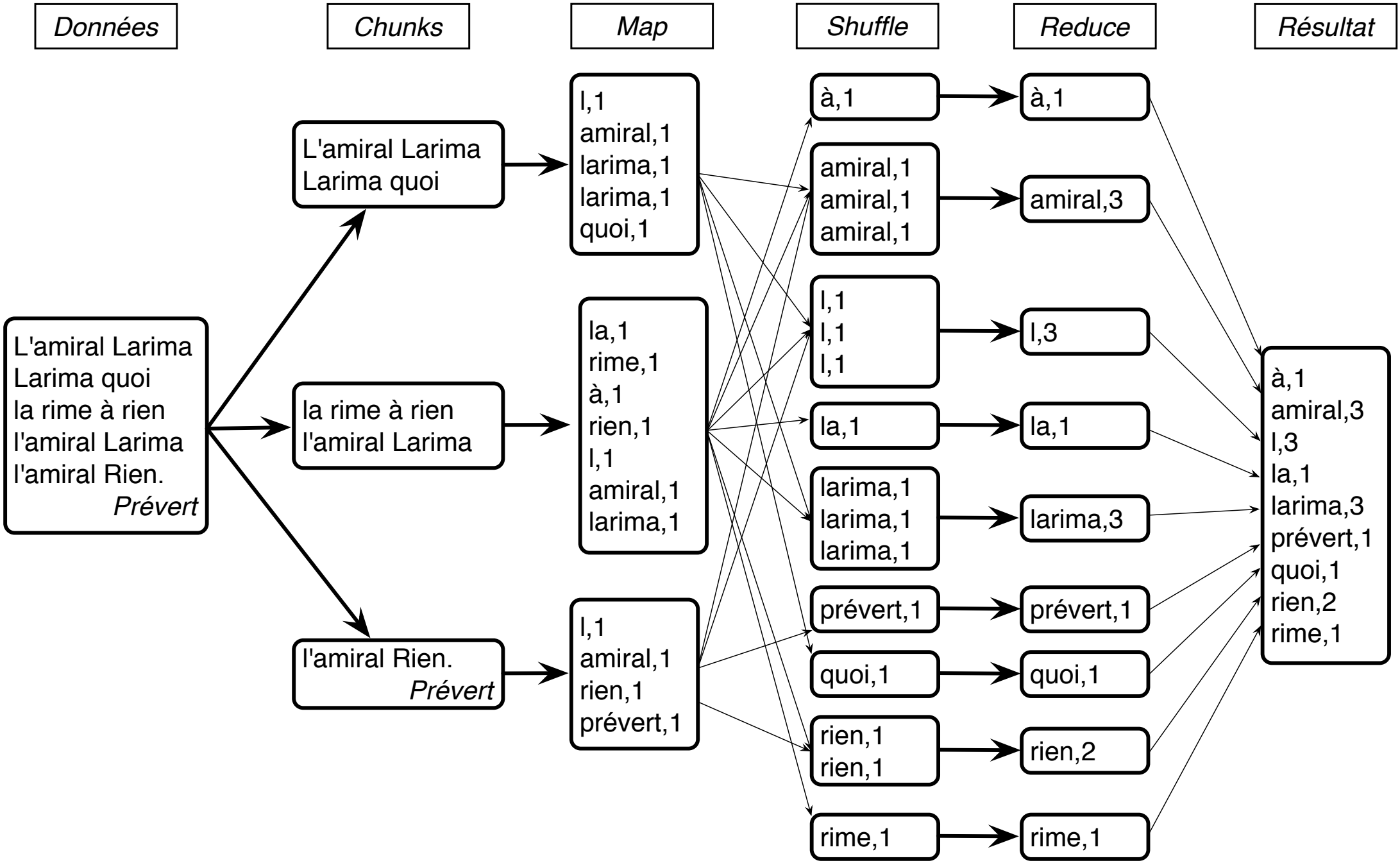
Exemple : indexage d'un corpus de texte

Les ouvriers comptent le nombre d'occurrences des mots de leur fragement, et produisent des listes triées (mot (*clé*),nb occurrence(*valeur*)), qui sont ensuite agrégées

Mise en œuvre Hadoop

Idée : amener les calculs aux données → la répartition des ouvriers suit celle des chunks


Chunks
Splits
Résultats (disque)
Résultat
envoi aux reducers
duplication
A
B
C
A'
B'
C'
A"
B"
C"
Map
Shuffle :
tri, partition
Shuffle
K1,1
K2,1
Kn,1
K1,<1,1…>
K2,<1,1…>
Kn,<1,1…>
Reduce
R1
R2
DataNode


Données
Chunks
Map
Shuffle
Reduce
Résultat
L'amiral Larima
Larima quoi
la rime à rien
l'amiral Larima
l'amiral Rien.
Prévert
L'amiral Larima
Larima quoi
la rime à rien
l'amiral Larima
l'amiral Rien.
Prévert
l,1
amiral,1
larima,1
larima,1
quoi,1
la,1
rime,1
à,1
rien,1
l,1
amiral,1
larima,1
l,1
amiral,1
rien,1
prévert,1
à,1
amiral,1
amiral,1
amiral,1
l,1
l,1
l,1
la,1
larima,1
larima,1
larima,1
prévert,1
quoi,1
rien,1
rien,1
rime,1
à,1
amiral,3
l,3
la,1
larima,3
prévert,1
quoi,1
rien,2
rime,1
à,1
amiral,3
l,3
la,1
larima,3
prévert,1
quoi,1
rien,2
rime,1

Terminologie Hadoop

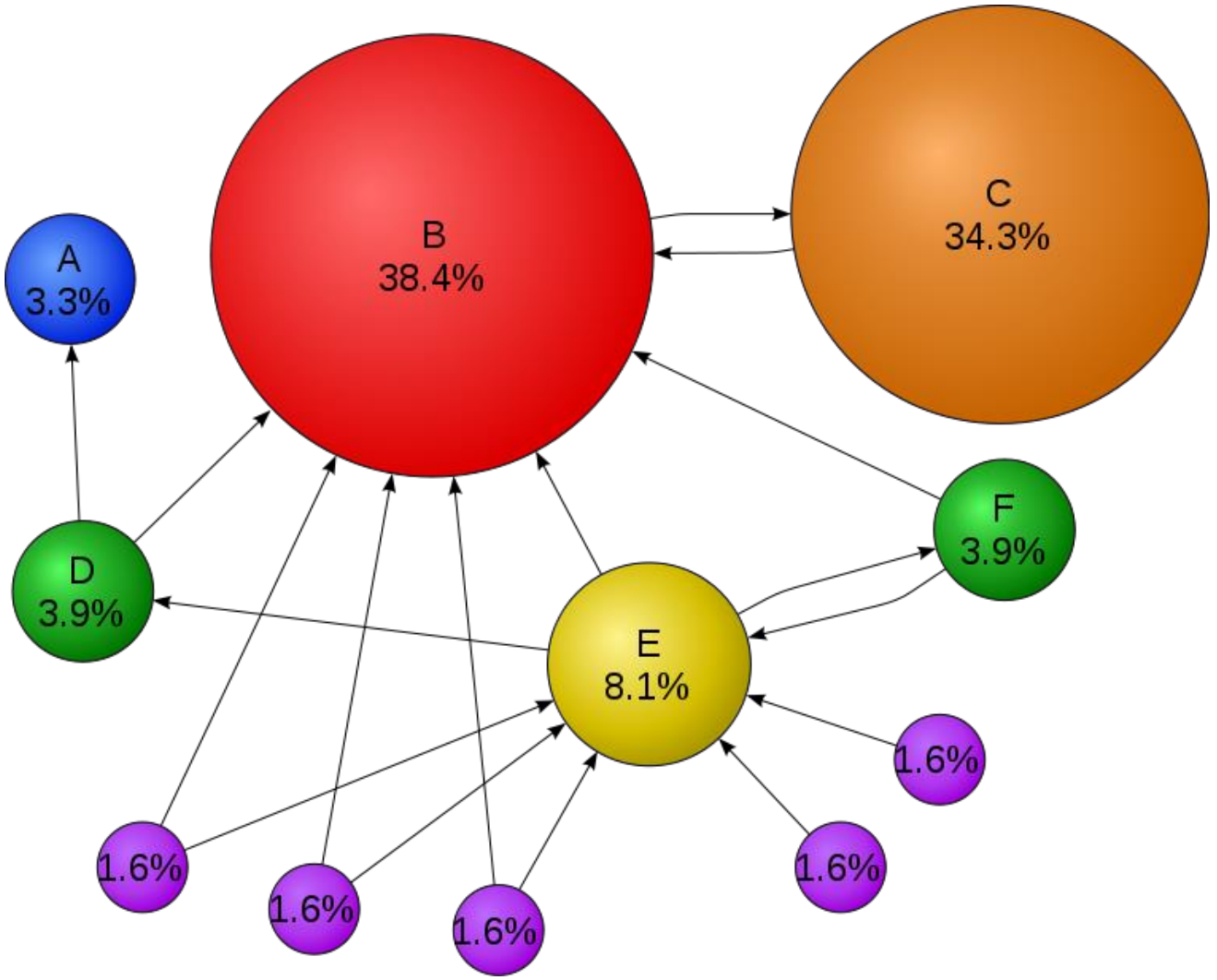
Fichiers

- *NameNode* : gère les métadonnées du SGF
- *DataNode* : mise en en œuvre des accès aux chunks sur un nœud

Exécution

- *RessourceManager* : gestion globale des ressources (allocation, supervision)
- *NodeManager* : gère les ressources sur un nœud
- *ApplicationMaster* : supervise l'exécution pour une application donnée

Design d'une solution MapReduce pour l'algorithme PageRank

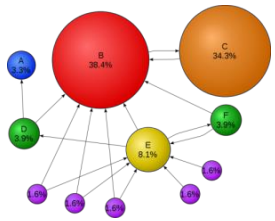


Qu'est-ce que PageRank?

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

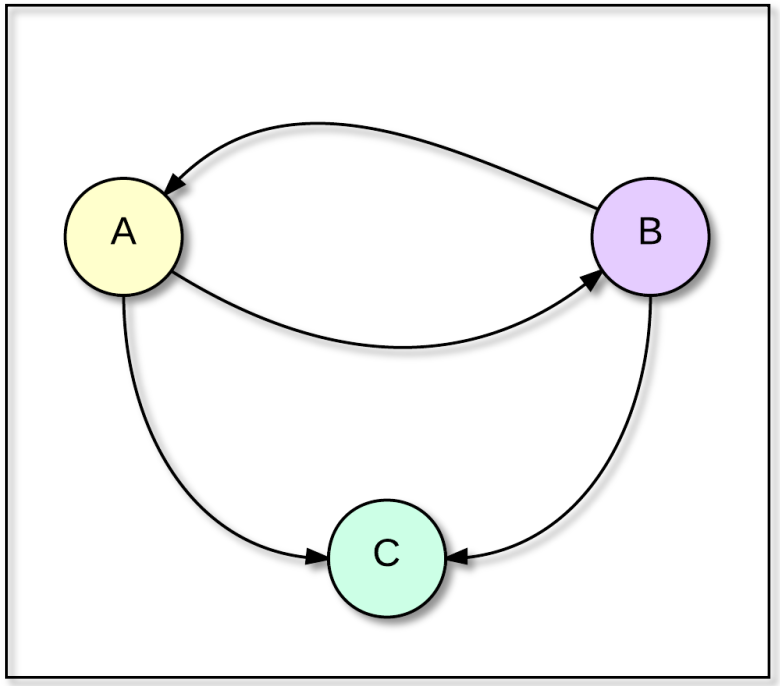
Initialisation: $PR(A) = PR(B) = PR(C) = 0.33$

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

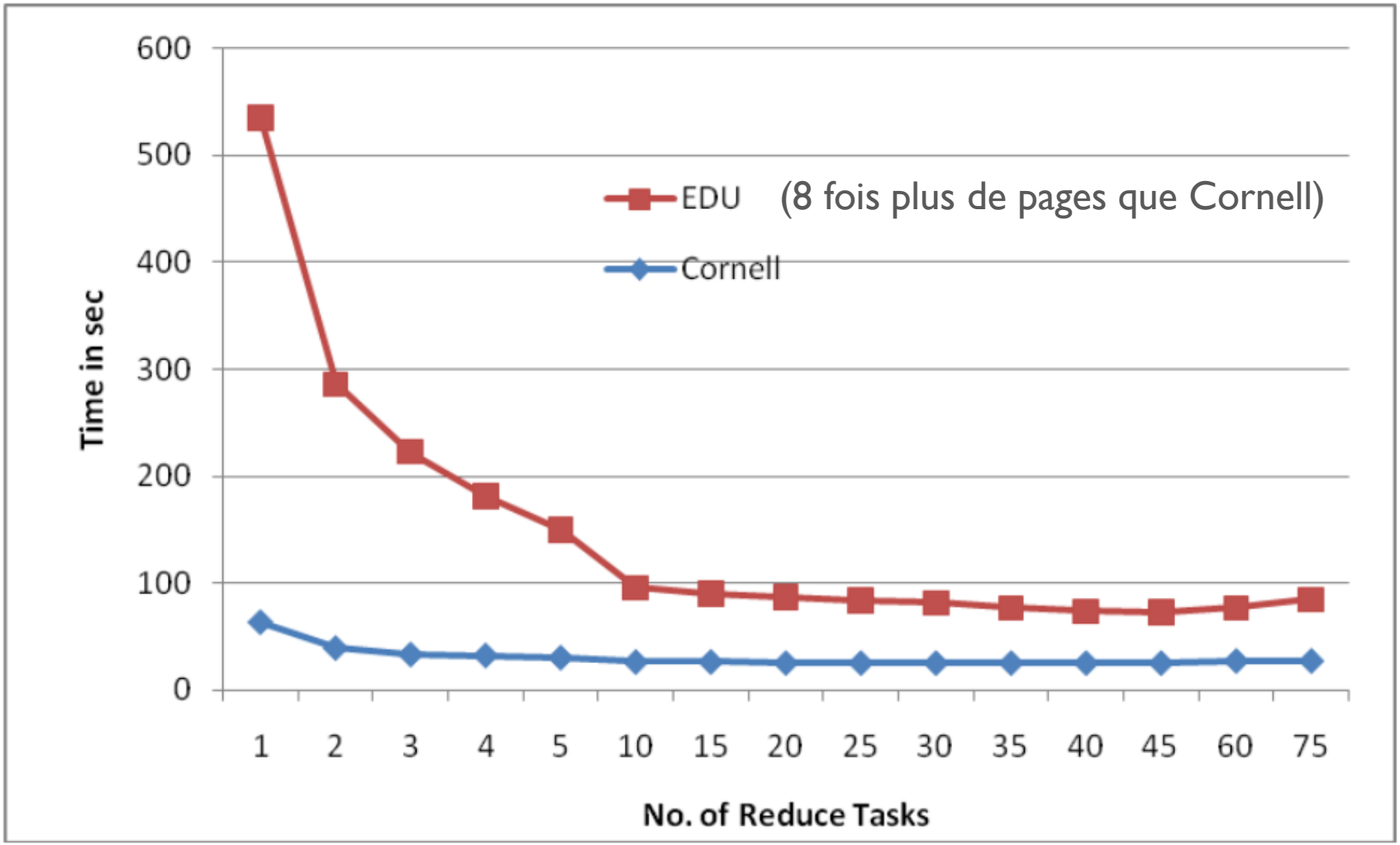
Table 1 : Details of the datasets used to test the page rank algorithm.

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		

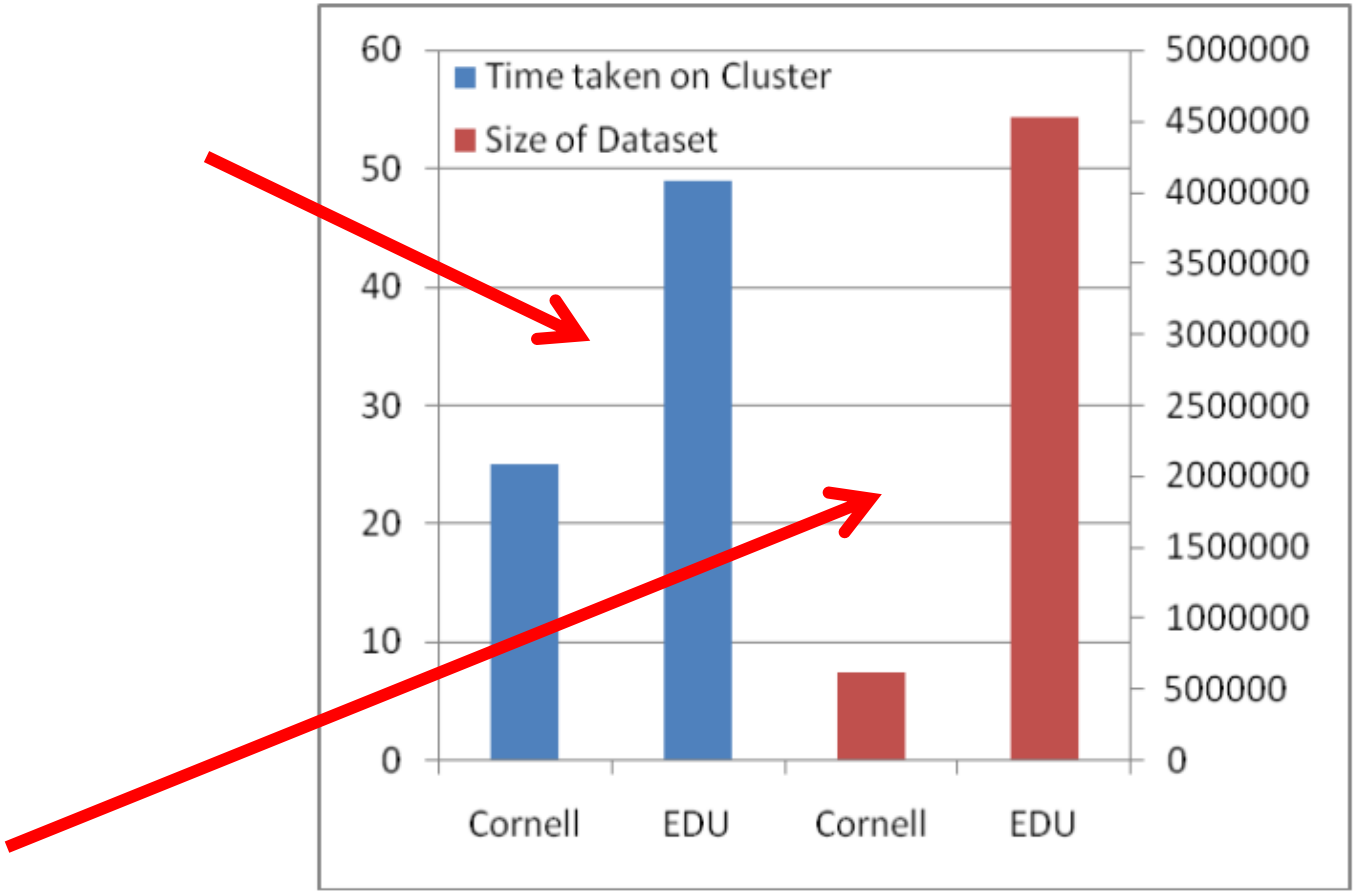
Table 2: Execution time taken by each of the three modules for 3 datasets.

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