

MapReduce

Algorithme de parallélisations des traitements

Khaled TANNIR
Doctorant CIFRE LARIS/ESTI



Présentation

Doctorant CIFRE LARIS / ESTI – Cergy-Pontoise

Consultant , Architecte Technique .NET- Groupe onePoint

Titre de la thèse:

**« Outils et services de fouilles de données
intégrés dans une grille sémantique »**

Directeur de thèse : M. Hubert KADIMA

Encadrants : Mme Maria MALEK, M. Cristian Dan VODISLAV

Sommaire

- Introduction à MapReduce
- L'algorithme MapReduce
- Mise en œuvre
- Avantages et inconvénients
- Perspectives

MapReduce

Introduction

Qu'est-ce que c'est ?

- Un *framework* de traitement distribué sur de gros volumes de données
- Un modèle de programmation parallèle conçu pour la scalabilité et la tolérance aux pannes
- *Map* et *Reduce* sont inspirés du langage fonctionnel *Lisp*
- Répartit la charge sur un grand nombre de serveurs et gère la distribution de données
- Conçu par *Google* et traite 20 Po de données par jour [1]

Qui utilise MapReduce ?

Google

- Construction des Index pour *Google Search*
- Regroupement des articles pour *Google News*

YAHOO!  hadoop

- Alimenter *Yahoo! Search* avec “*Web map*”
- Détection de Spam pour *Yahoo! Mail*

facebook

- Data mining (fouille de données)
- Détection de Spam

• ...



Source Microsoft MSDN

La 'Recherche' l'utilise aussi

- Laboratoires / Chercheurs
 - Analyse d'images Astronomique
 - Bioinformatique
 - Simulation métrologique
 - Simulation physiques
 - Machine Learning
 - Statistiques
 - <Mon application ici>



Avant MapReduce...

- Pour un développeur Il est difficile de :
 - Traiter de grands volumes de données
 - Gérer de milliers de processeurs
 - Paralléliser et distribuer des traitements
 - Ordonnancement des entrées / sorties
 - Gérer la tolérance aux pannes
 - Surveiller des processus
- MapReduce fournit tout ceci, facilement!

MapReduce, pour quelle problématique ?

- Itérer sur un grand nombre d'enregistrements
- Extraire quelque chose ayant un intérêt de chacun d'eux

Map



- Regrouper et trier les résultats (intermediaires)
- Agréger ces résultats
- Générer le résultat final

Reduce

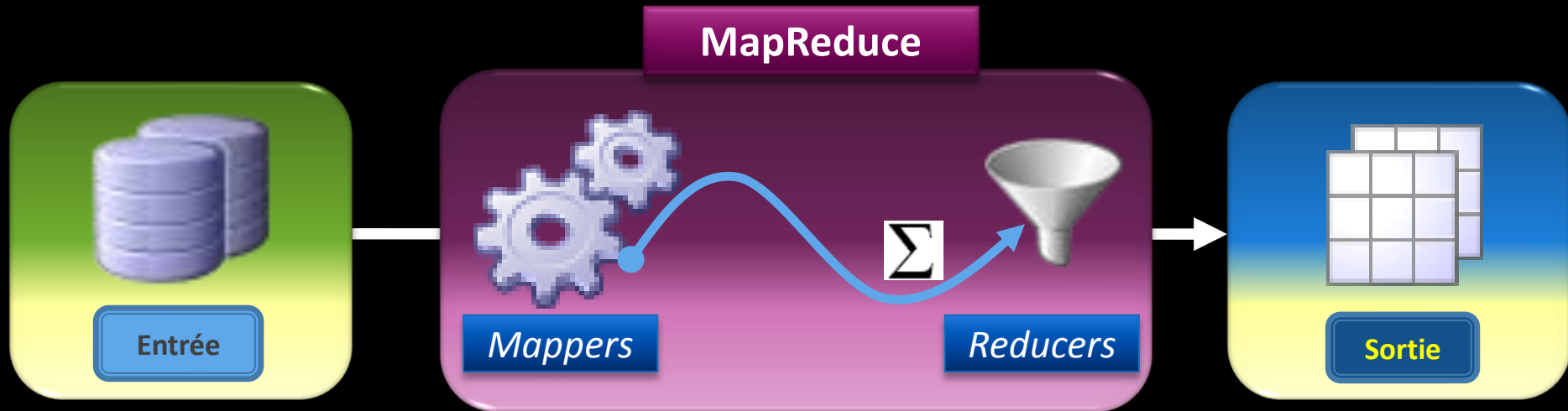


Fournir une abstraction fonctionnelle de ces deux opérations

L'algorithme MapReduce

Modèle de programmation

Fonction globale de MapReduce



Principes de base de l'algorithme

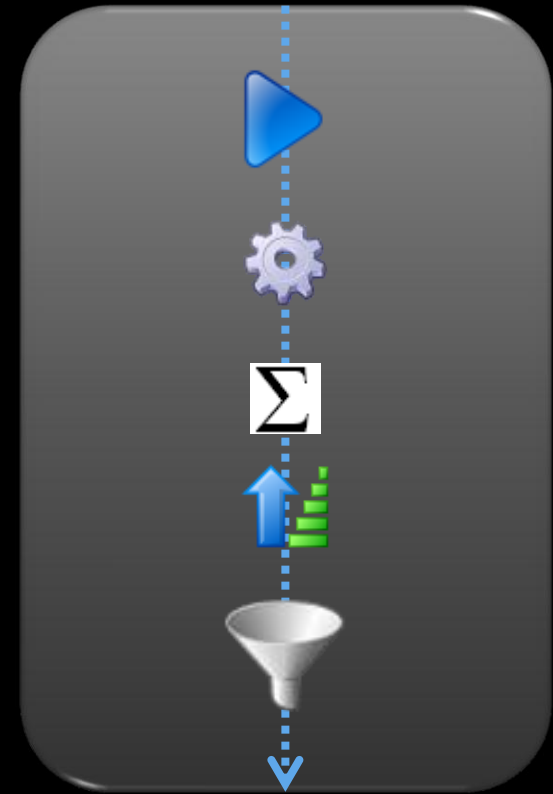
- Une tâche est divisée en deux ou plusieurs sous-tâche
 - Chaque sous-tâche est traitée indépendamment, puis leurs résultats sont combinés
- 3 opérations majeures : ***Split, Compute et Join***
- C'est le principe de « *Diviser pour Reigner* » avec une structure pseudo-hiérarchique

Modèle de programmation

- L'utilisateur fournit deux fonctions
 - **Map**
 - Prend en entrée un ensemble de « **Clé, Valeurs** »
 - Retourne une liste intermédiaire de « **Clé1, Valeur1** »
 - **Map(key,value) → list(key1,value1)**
 - **Reduce**
 - Prend en entrée une liste intermédiaire de « **Clé1, Valeur1** »
 - Fournit en sortie un ensemble de « **Clé1,Valeur2** »
 - **reduce(key1, list(value1)) → value2**

Les phases de MapReduce

- Initialisation
- **Map**
- **Shuffle (regroupement)**
- **Sort (tri)**
- **Reduce**



MapReduce: L'étape "Map"

Entrée

Paire « **Key-value** »

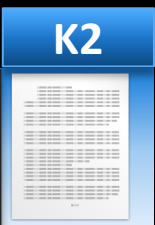
V1

K1



V2


K2



...

Vn

Kn



Map(doc-Id, doc-content)

Map

Map

Intermédiaire

Paires « **key-value** »

k₁



v₁

k₂



v₁

k₁



v₁

...

k₁



v₁

(word, wordcount-in-a-doc)

MapReduce: L'étape "Reduce"

Intermédiaire

Paires « **key-value** »



...



(word, wordcount-in-a-doc)

Grouper

Regroupement



...



(word, list-of-wordcount)

~ SQL Group by

Reduce

Reduce

Sortie



...



(word, final-count)

~ SQL aggregation

Le ‘Maître’ MapReduce (Master)

- Coordonne l’exécution des “unités de travail” (*Workers*)
 - Attribue aux unités de travail les tâches “**map**” et “**reduce**”
- Gère la distribution des données
 - Déplace les ‘workers’ vers les données
- Gère la synchronisation
 - Regroupe, trie, et réorganise les données intermédiaires
 - Détecte les défaillances des *unités de travail* et relance la tâche

Mise à l'œuvre de MapReduce

Exemple de fonctionnement

“Word Count” avec MapReduce

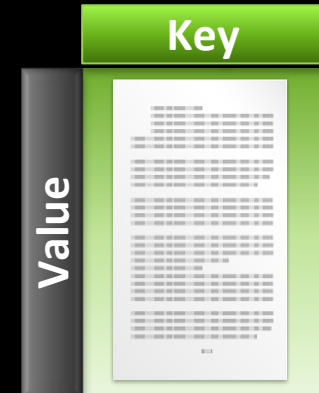
mapFunc(String key, String value):

// key: nom du document;

// value: contenu du document

for each word **w** in value:

EmitIntermediate (w, 1)



reduceFunc(String key, Iterator values):

// key: un mot;

// value: une liste de valeurs

result = 0

for each count **v** in values:

result += v

EmitResult(result)



Structure de données du 'Worker'

- Un 'Worker' est une "unité de travail" qui possède trois états :
 - **idle**, **in-progress**, **completed**
 - **Idle** indique qu'un *worker* est disponible pour une nouvelle planification
 - **Completed** indique la fin d'un traitement, le *worker* informe le *Master* de la taille, de la localisation des ses fichiers intermédiaires
 - **In-progress** indique q'un traitement est en cours
- Les 'reducers' sont informés des états des *workers* par le Master

La gestion des données

- Les données en Entrée et en Sortie sont stockées sur un system de fichiers distribués
- Les données sont stockées au plus proche de leur source
- Les données intermédiaires sont stockées sur le système de fichier local des unités « **map** » et « **reduce** »
- Les données en sortie représentent souvent une entrée pour une autre unité MapReduce

La gestion des défaillances

1/2

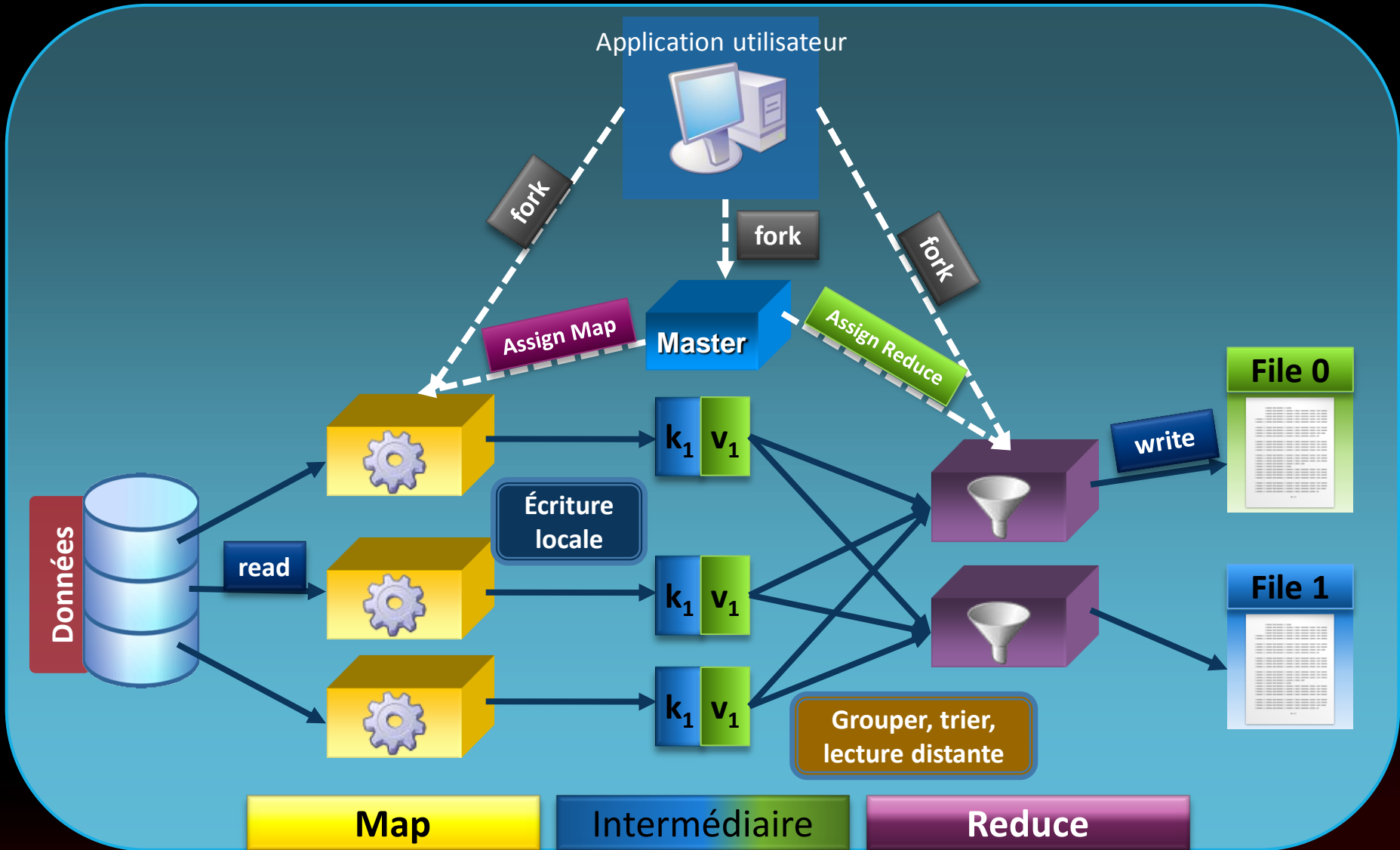
- Basée sur un mécanisme de réexécution
- Le *Master* 'ping' régulièrement les unités « **map** » et « **reduce** »
- En cas de défaillance d'une unité « **map** »
 - Les tâches complètes ou en cours d'exécution seront réinitialisées (état: **in-progress**, **completed**)
 - Les tâches seront placées dans de nouvelles unités de travail dans de nouveaux nœuds du système de fichiers

- En cas de défaillance d'une unité « **reduce** »
 - Les tâches complètes (état: **completed**) ne sont pas relancées
 - Uniquement les tâches en cours d'exécution seront réinitialisées (état: **in-progress**) et relancées sur un autre nœud du système de fichiers distribués
- En cas de défaillance du « **Master** »
 - Les tâches MapReduce seront abandonnées et le client (utilisateur) est notifié

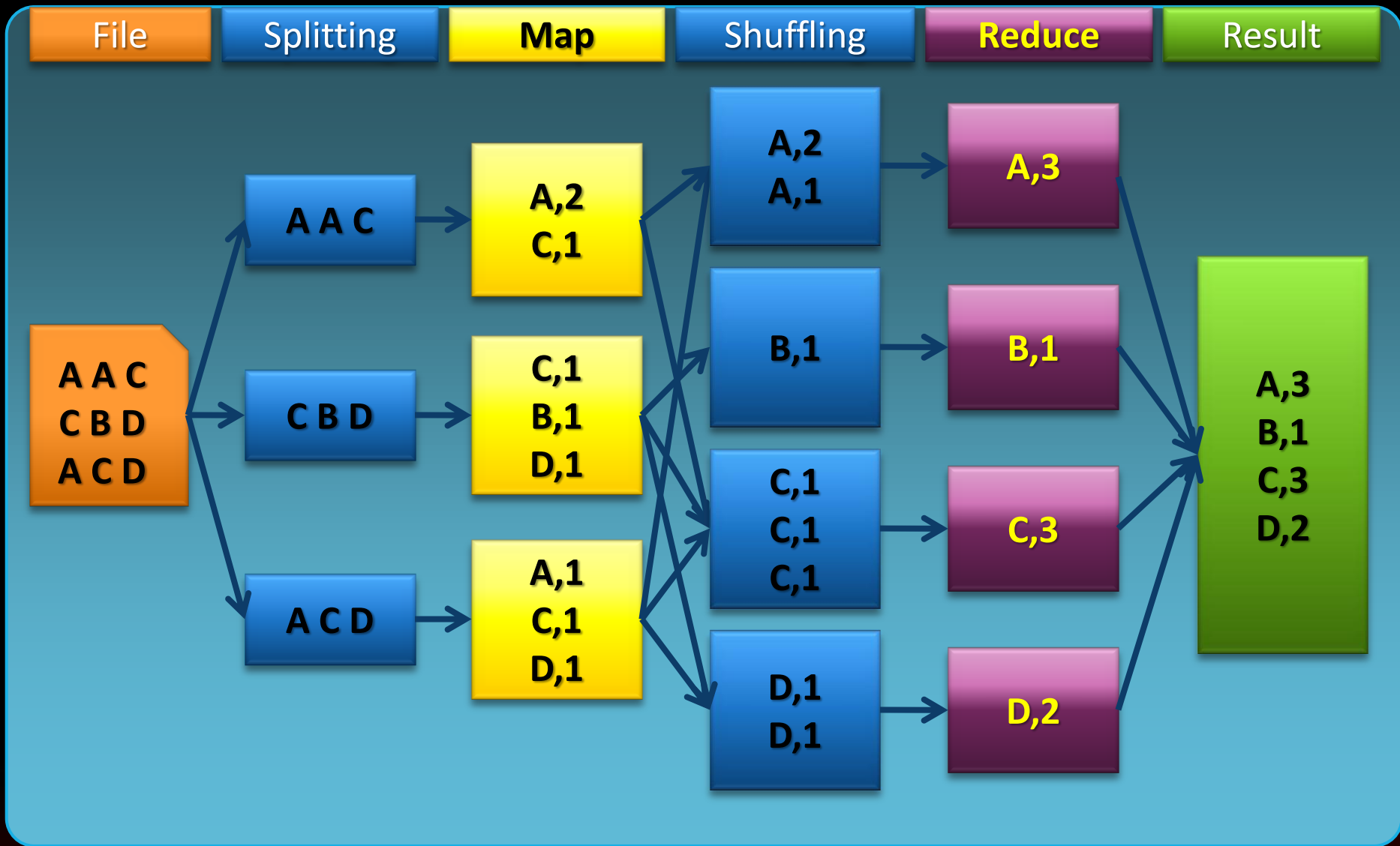
Les mécanismes d'optimisations

- Mapreduce implémente plusieurs mécanismes d'optimisations
 - Optimisation de la bande passante réseau
 - Utilise un mécanisme de spéculation pour placer des tâches sur des nœuds différents du File System
- Les '*Combiner*' sont utilisés pour réduire le volume de données transmis entre **map** et **reduce**.
 - Utilise un mécanisme similaire à la fonction *reduce*

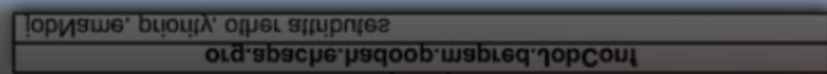
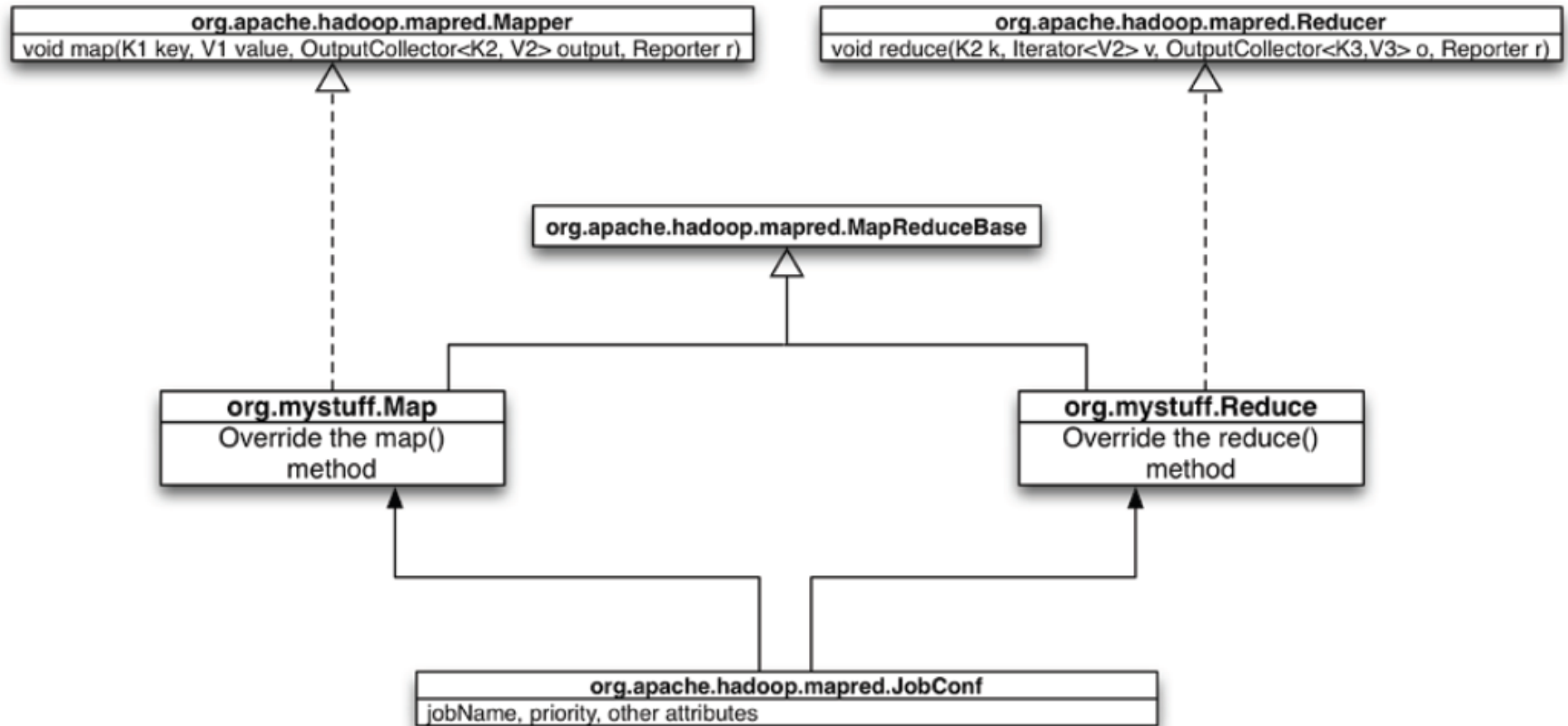
Mise en œuvre de MapReduce



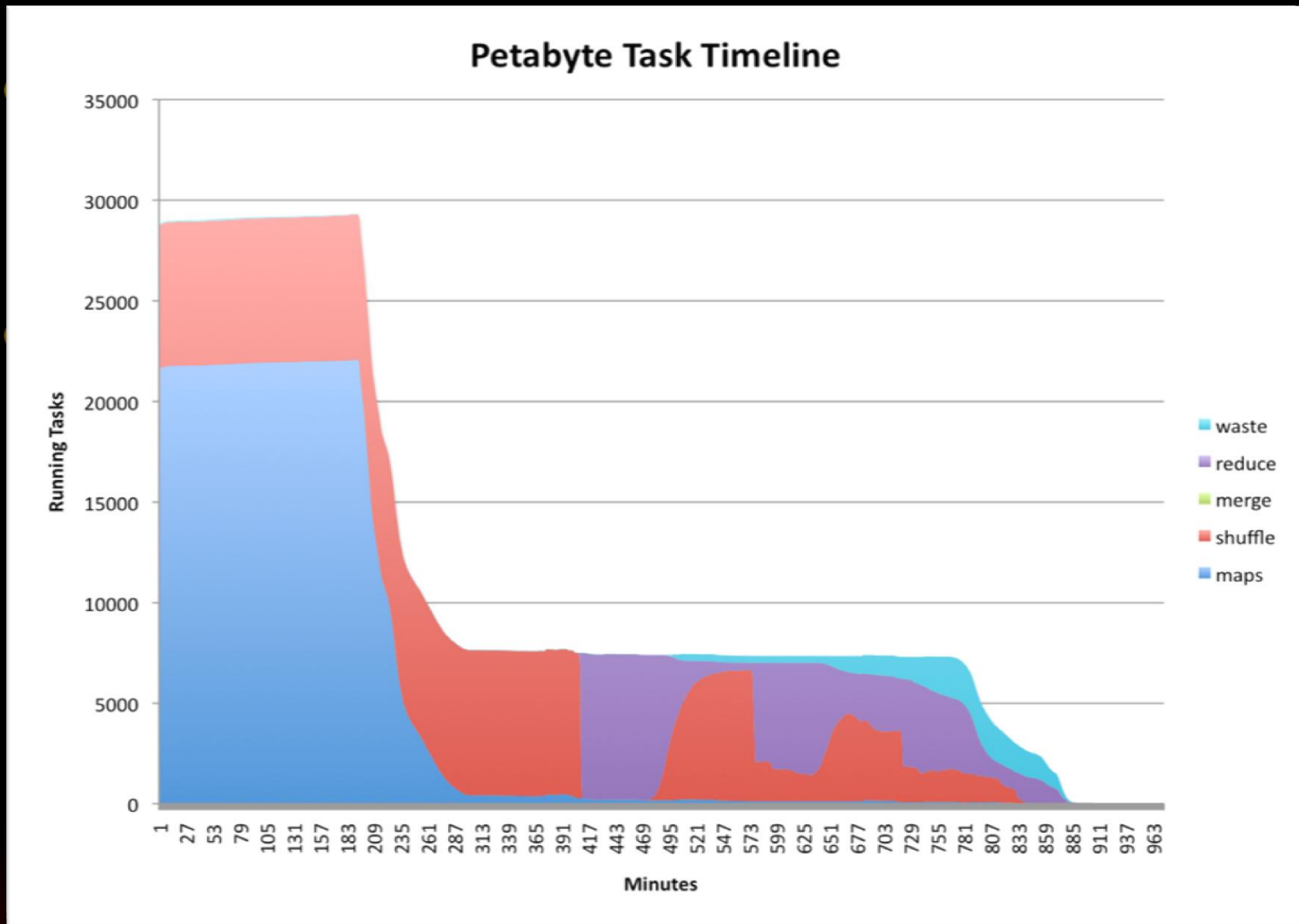
Exemple de mise en œuvre



Les interfaces Java de MapReduce



Les performances de MapReduce



Exemples d'applications

- Calculer la taille de plusieurs milliers de documents
- Trouver le nombre d'occurrence d'un pattern dans un très grand volume de données
- Classifier des très grands volumes de données provenant des paniers d'achats de clients
- ...

Avantages et Inconvénients

Quand utiliser MapReduce

Avantages de MapReduce

- Fourni une abstraction totale des mécanismes de parallélisations sous-jacents
- Peu de tests sont nécessaires. Les bibliothèques MapReduce ont déjà été testées et fonctionnent correctement
- L'utilisateur se concentre sur son propre code
- Largement utilisé dans les environnements de *Cloud Computing*

Inconvénients de MapReduce

- Une seule entrée pour les données
- Deux primitives de haut-niveau seulement
- Le flux de données en deux étapes le rend très rigide
- Le système de fichiers distribués (HDFS) possède une bande passante limitée en entrée / sortie
- Les opérations de tris limitent les performances du Framework (implémentation Hadoop)

Quand utiliser MapReduce ?

- MapReduce constitue une bonne solution lorsqu'il y a beaucoup de données:
 - En entrée (ex: calculs statistiques)
 - De fichiers intermediaires (ex: phrase tables)
 - En Sortie (ex: webcrawls)
 - Peu de synchronisation est nécessaire entre les unités de travail

Quand ne pas utiliser MapReduce ?

- L'utilisation de MapReduce peut être problématique
 - Des traitements "Online" sont nécessaires
 - L'utilisation d'algorithmes du type Perceptron
 - Les opérations individuelles **map** ou **reduce** sont extrêmement coûteuses en calcul
 - De grands volumes de données partagées sont nécessaires

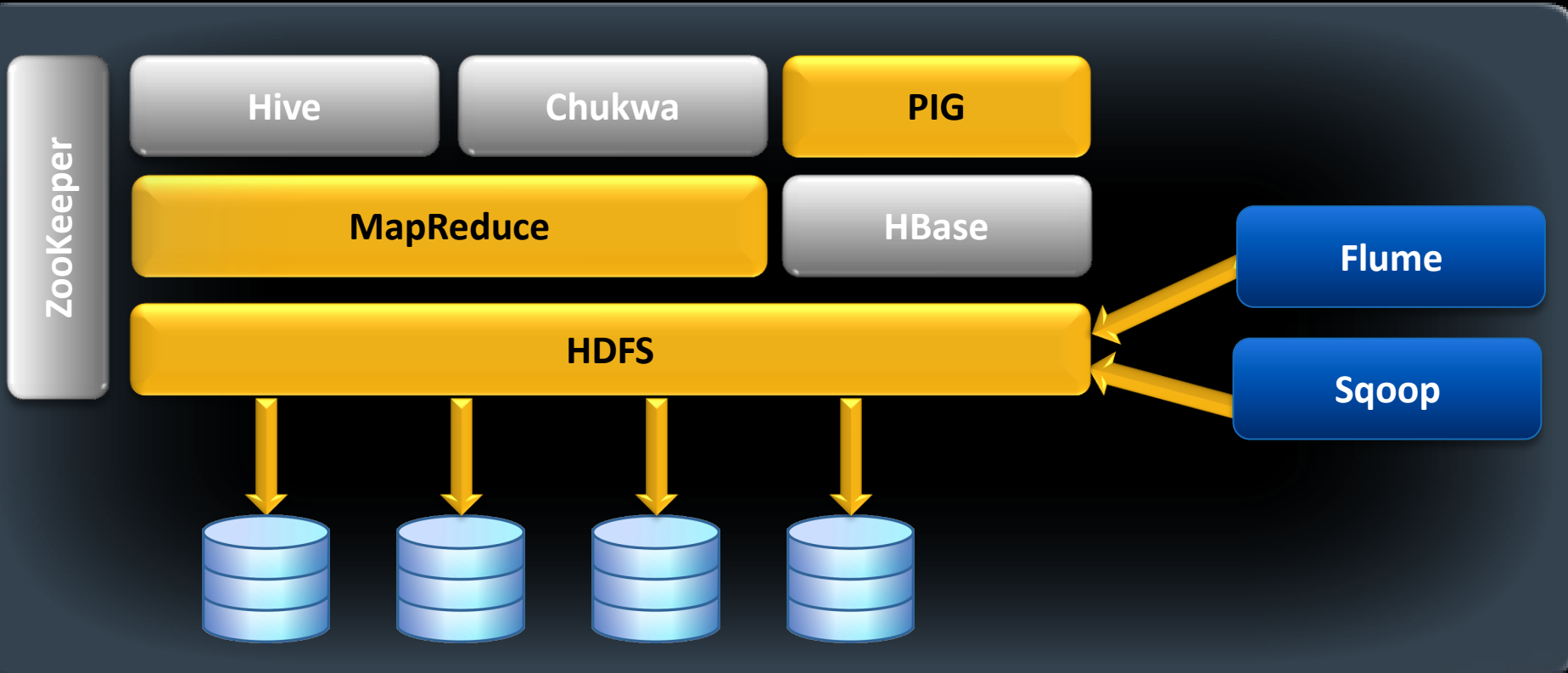
Perspectives

MapReduce et mon travail futur

Les différentes implémentations

- **Hadoop** (Yahoo) constitue un modèle équivalent et étendu
- **DryadLinQ** (Microsoft) est une approche un peu plus générique
- **MapReduce-Merge** qui étend le framework avec la possibilité de fusionner des résultats
- Elastic (Amazon)
- Et d'autres ...

Les composants Hadoop en un mot

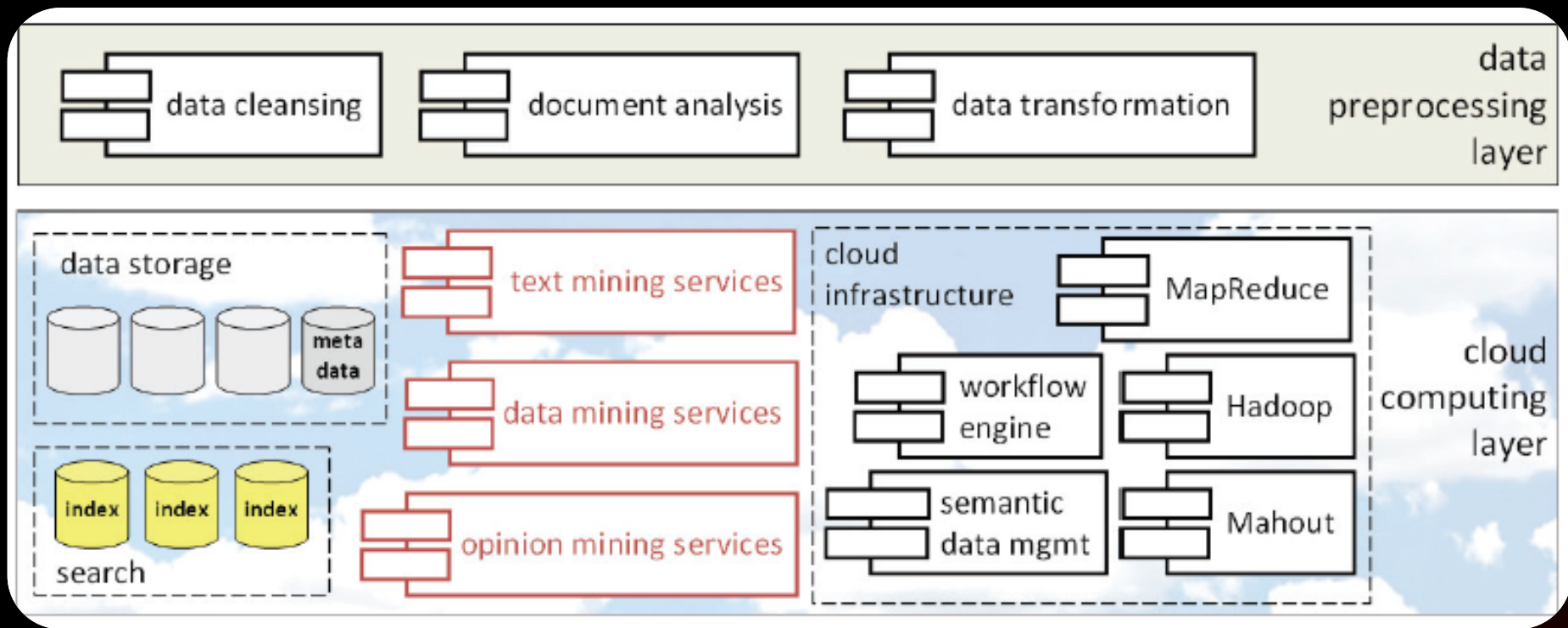


MapReduce et mon travail de recherche

- Implémenter MapReduce au niveau PaaS du Cloud
- Utiliser *Cassandra*¹ (ou autre) pour combler les limites du système de fichier de MapReduce
- L'adaptation d'un (ou plusieurs) algorithmes de classification comme *Apriori* et exploiter MapReduce
- ...

1 – Un système open-source de stockage de fichiers distribués géré par Apache

La plateforme cible de mon travail

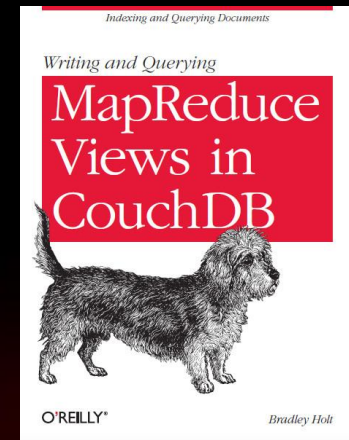
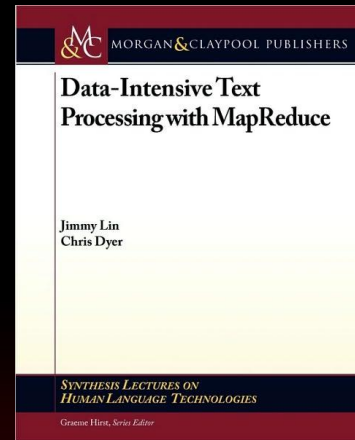
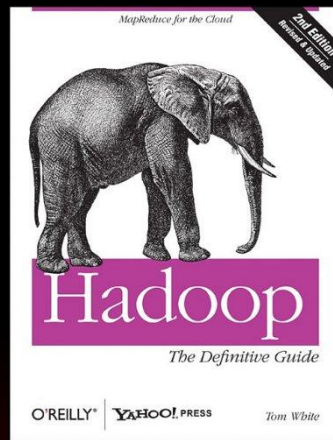
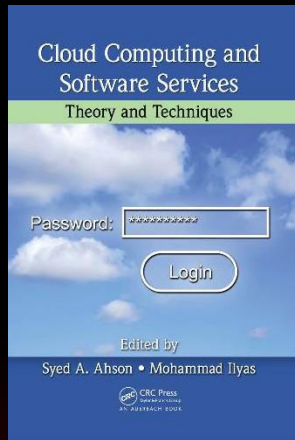


Conclusion

- MapReduce est un modèle de programmation facile d'utilisation
- Il est robuste et permet de traiter de très gros volume de données
- A été mis dans le domaine public avec l'implémentation *Hadoop* de *Yahoo*
- Plusieurs projets universitaires ont permis de l'améliorer

Références : Ouvrages

- [1] Cloud Computing and Software Services Theory and Techniques— CRC Press 2011- Syed Ahson, Mohammad Ilyas – pages 93-137
- [2] Hadoop The Definitive Guide – O'Reily 2011 – Tom White
- [3] Data Intensive Text Processing with MapReduce – Morgan & Claypool 2010 – Jimmy Lin, Chris Dyer – pages 37-65
- [4] Writing and Querying MapReduce Views in CouchDB – O'Reily 2011 – Brandley Holt – pages 5-29



Références : URLs et Publications

[5] Jeffrey Dean and Sanjay Ghemawat, « [MapReduce: Simplified Data Processing on Large Clusters](#) » OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[6] Owen O'Malley, « [TeraByte Sort on Apache Hadoop](#) » , Yahoo!, May 2008

[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung « [The Google File System](#)« , 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

Rapports:

[8] Dzone Refcardz (<http://refcardz.dzone.com/>) N° 117, N° 133

[9] ERCIM NEWS- Issue 83 - October 2010 (<http://ercim-news.ercim.eu/images/stories/EN83/EN83-web.pdf>)

URLs:

[1] <http://labs.google.com/papers/mapreduce-osdi04-slides/index-auto-0002.html>

[2] <http://hadoop.apache.org>

[3] <http://fr.wikipedia.org/wiki/MapReduce>

[4] http://developer.yahoo.com/blogs/hadoop/posts/2009/05/hadoop_sorts_a_petabyte_in_162/

[5] <http://sortbenchmark.org/> : records de tri de données depuis 1987

Questions?

Merci de votre attention