MIF18 : Bases de Données Avancées

Framework MapReduce & Pig : traitement de données à l'échelle du Web

romuald.thion@univ-lyon1.fr

http://liris.cnrs.fr/~ecoquery/dokuwiki/doku.php?id=enseignement:bdav:start

Outline

Motivation originale : Google

Jeffrey Dean, Sanjay Ghemawat : *MapReduce : simplified data processing on large clusters* OSDI, 2004

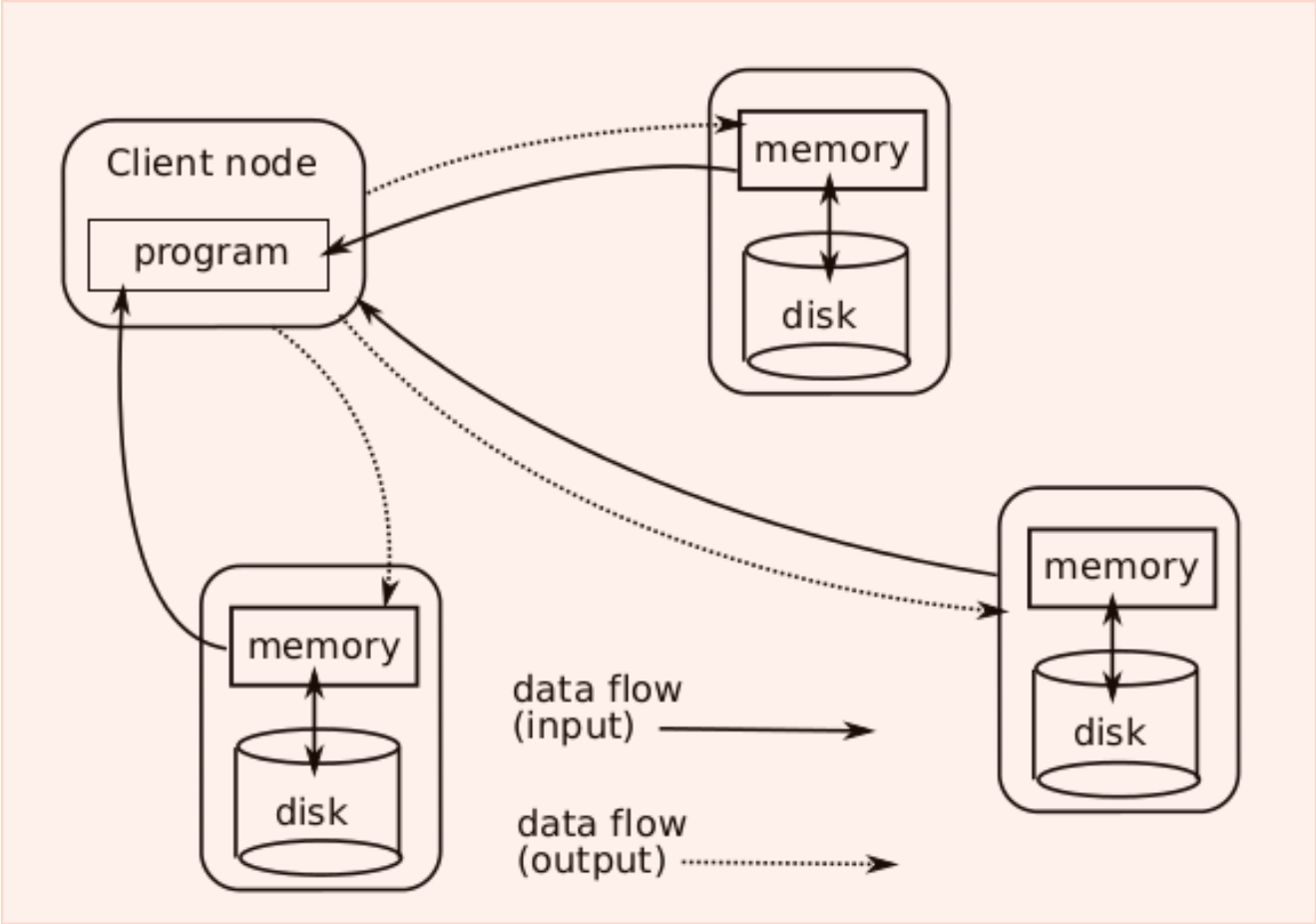
Contexte applicatif

- Famille de traitement *simples* :
  index inversé, statistiques, requêtes ou mots fréquents
- Sur de *très grands* volumes de données :
  page web, log d'accès, documents

Les challenges

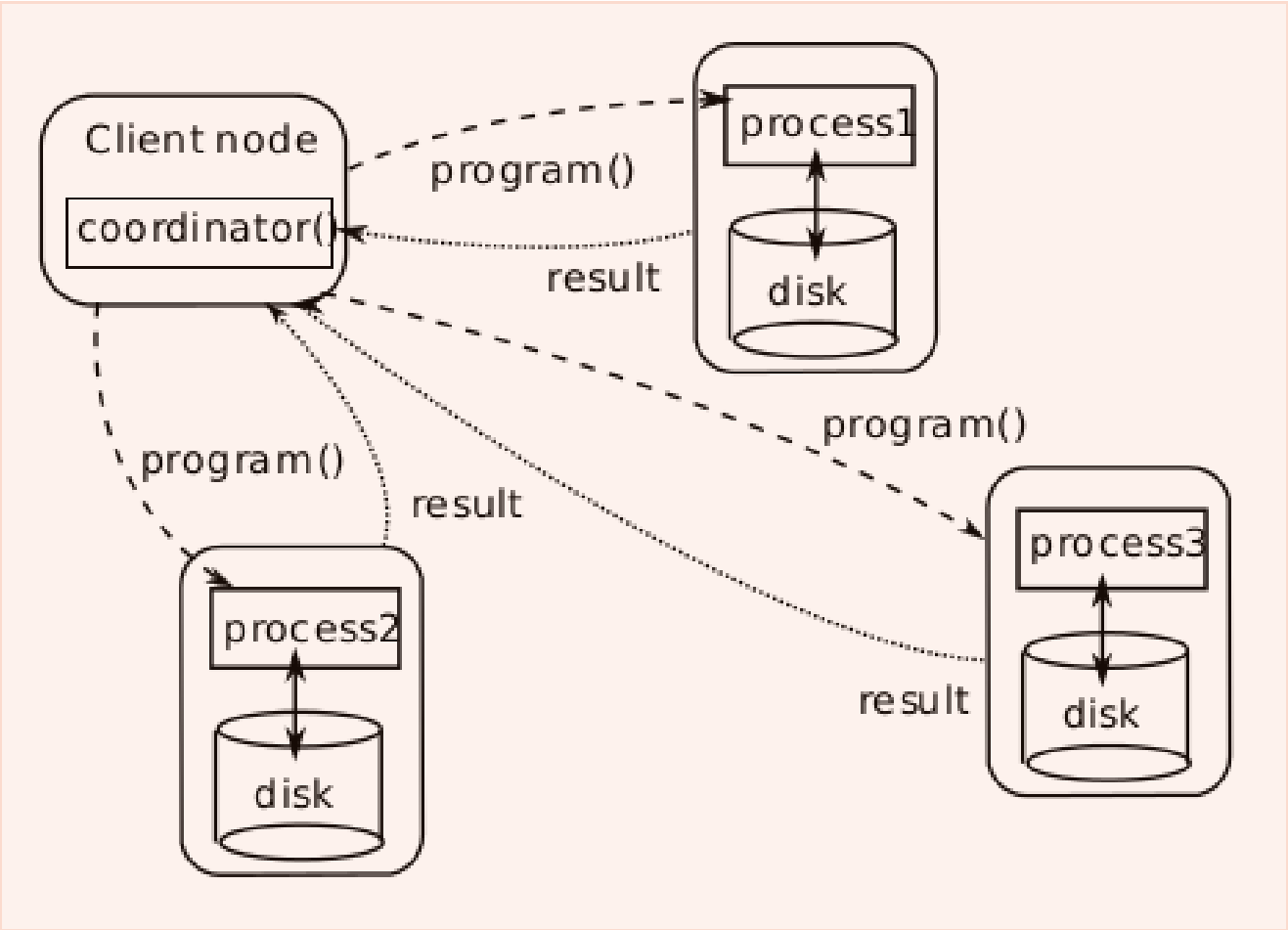
- Distribution des données
- Parallélisation des calculs
- Gestion des pannes
- Réduction des coûts (cluster de PCs)

Distribution des données



Coût élevé des transferts réseau (bottleneck)

Distribution des données *et* des calculs



Gérer les jobs pour qu'ils s'exécutent au plus proche des données

Hadoop : la référence

Stack Apache Hadoop

- *Hadoop Common* : The common utilities that support the other Hadoop modules.
- *Hadoop Distributed File System* (HDFS) : A distributed file system that provides high-throughput access to application data.
- *Hadoop YARN* : A framework for job scheduling and cluster resource management.
- *Hadoop MapReduce* : A YARN-based system for parallel processing of large data sets.

Hadoop : la référence

Qui utilise Hadoop ?

Hadoop : la référence

Projets Apache connexes à Hadoop

- *Ambari* : A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters
- *Chukwa* : A data collection sys. for managing large distributed systems.
- *ZooKeeper* : A coordination service for distributed applications.
- *Cassandra* : A scalable multi-master DB with no single points of failure.
- *HBase* : A scalable, distributed database that supports structured data storage for large tables.
- *Hive* : A data warehouse infrastructure that provides data summarization and ad hoc querying.
- *Mahout* : A scalable machine learning and data mining library.
- *Pig* : A high-level data-flow language and execution framework for parallel computation.

Hadoop Distributed File System

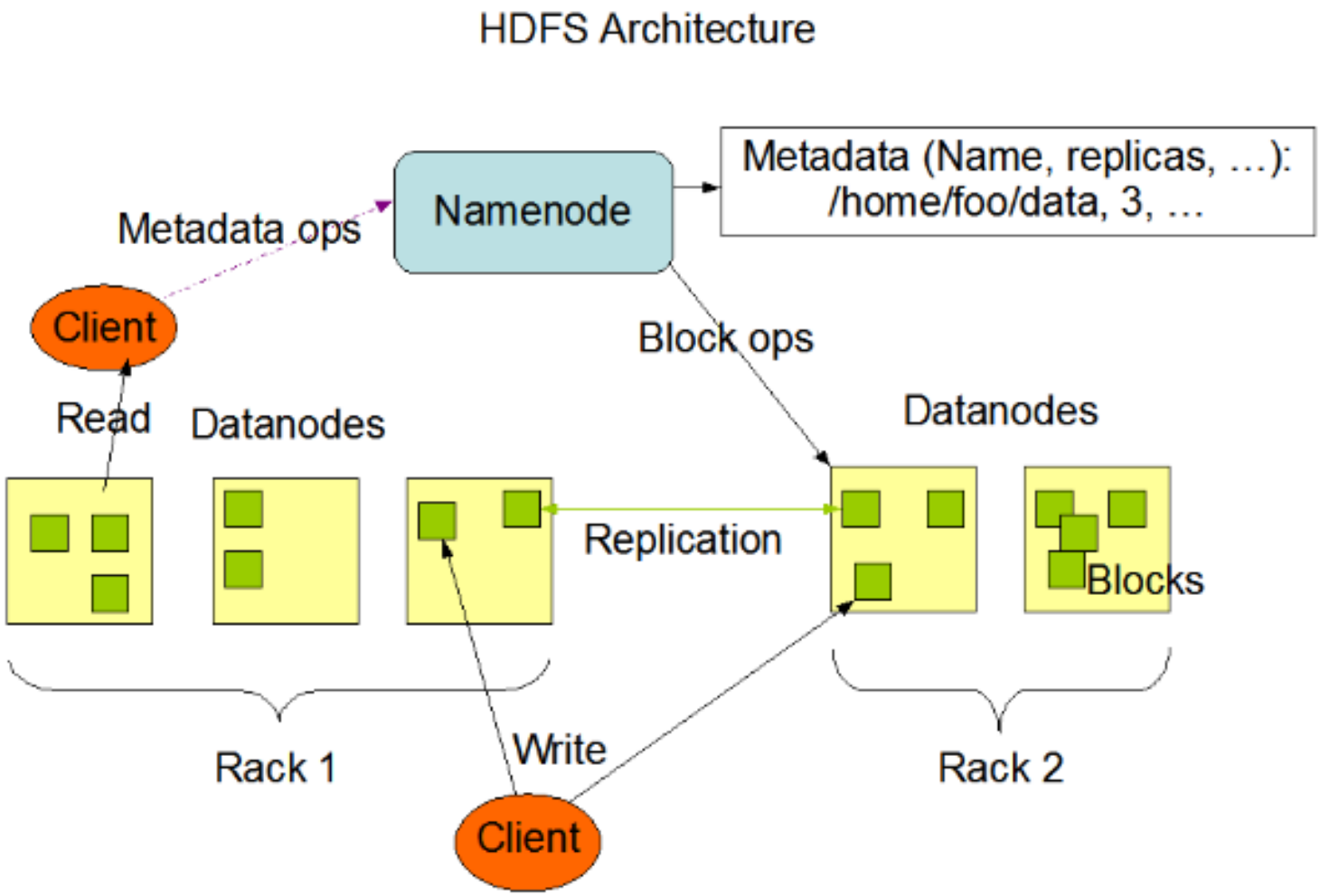
HDFS = la couche d'accès aux données de Hadoop.

- Peut être remplacée par une autre
  e.g., HBase, Cassandra, SGBD-R pour ApacheDB
- Nœud du cluster Hadoop = HDFS + MapReduce

Principes et objectifs

1. La panne matérielle est la norme, pas l'exception
2. Le batch/stream est privilégié à l'interactif (e.g., édition de doc)
3. Doit fonctionner avec des giga ou tera de données
4. Modèle simplifié de cohérence : *write-once-read-many*
5. « *Moving Computation is Cheaper than Moving Data* »
6. Portabilité entre OS

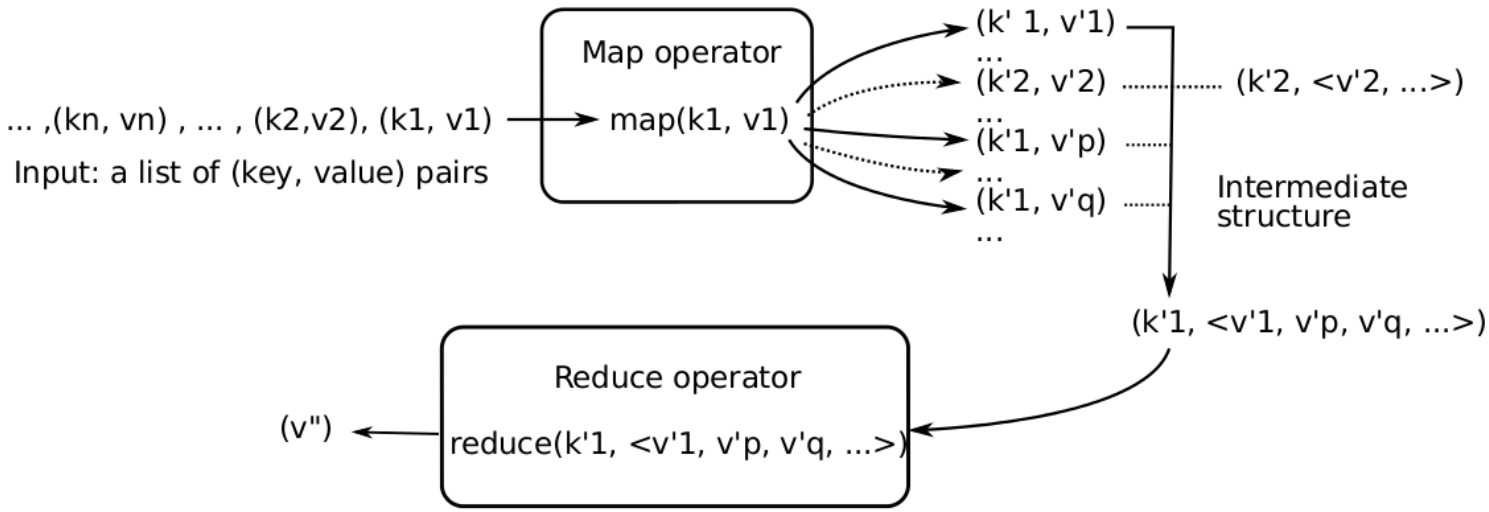
Hadoop Distributed File System



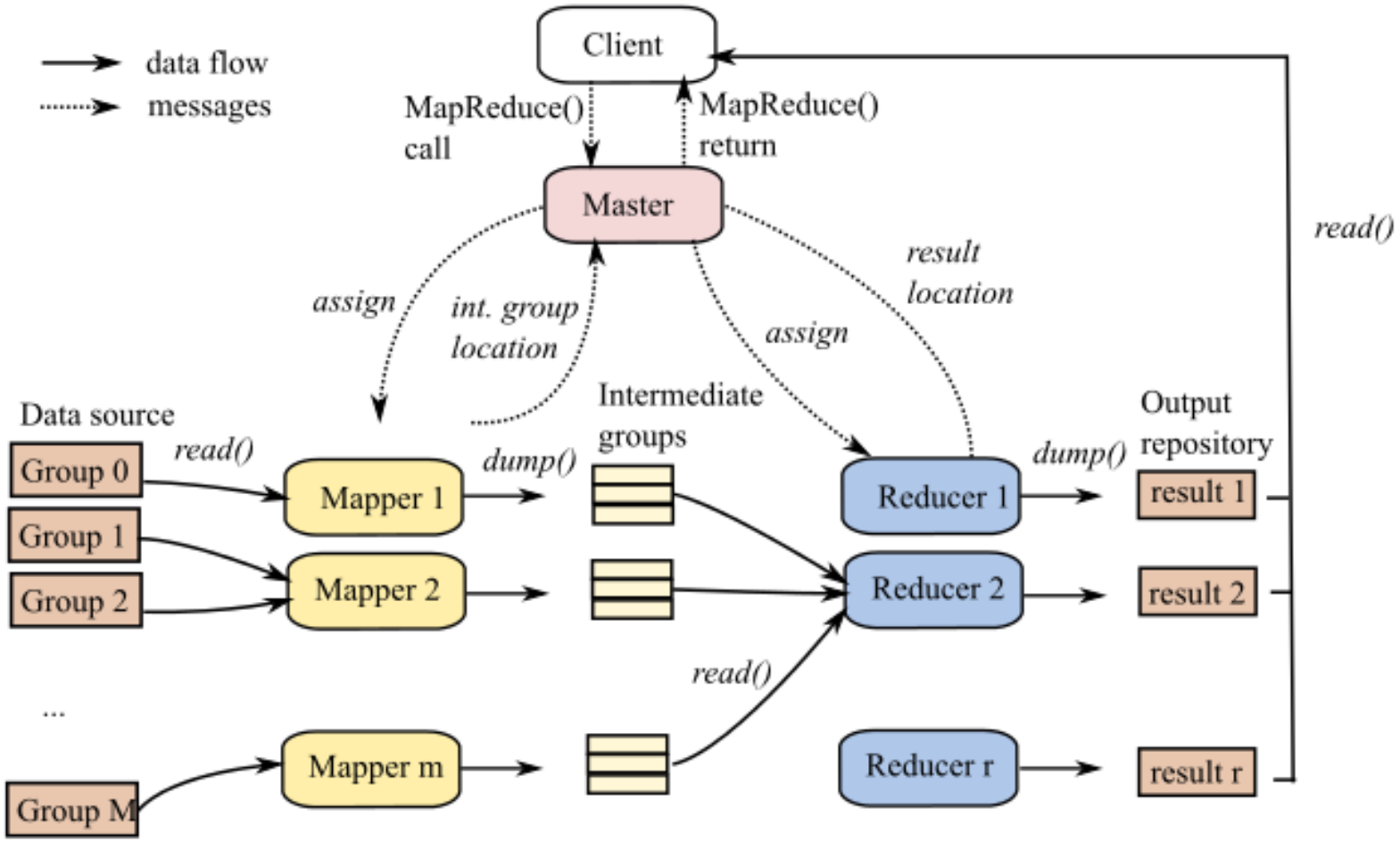
Pipeline MapReduce

Composition de trois opérations

1. Map : applique une fonction à une collection
   Mapper : nœud qui exécute une partie de Map
2. Sort/Group/Shuffle/ : réorganisation automatique des résultats intermédiaires
3. Reduce : agrège les résultats intermédiaires
   Reducer : nœud qui exécute une partie de Reduce

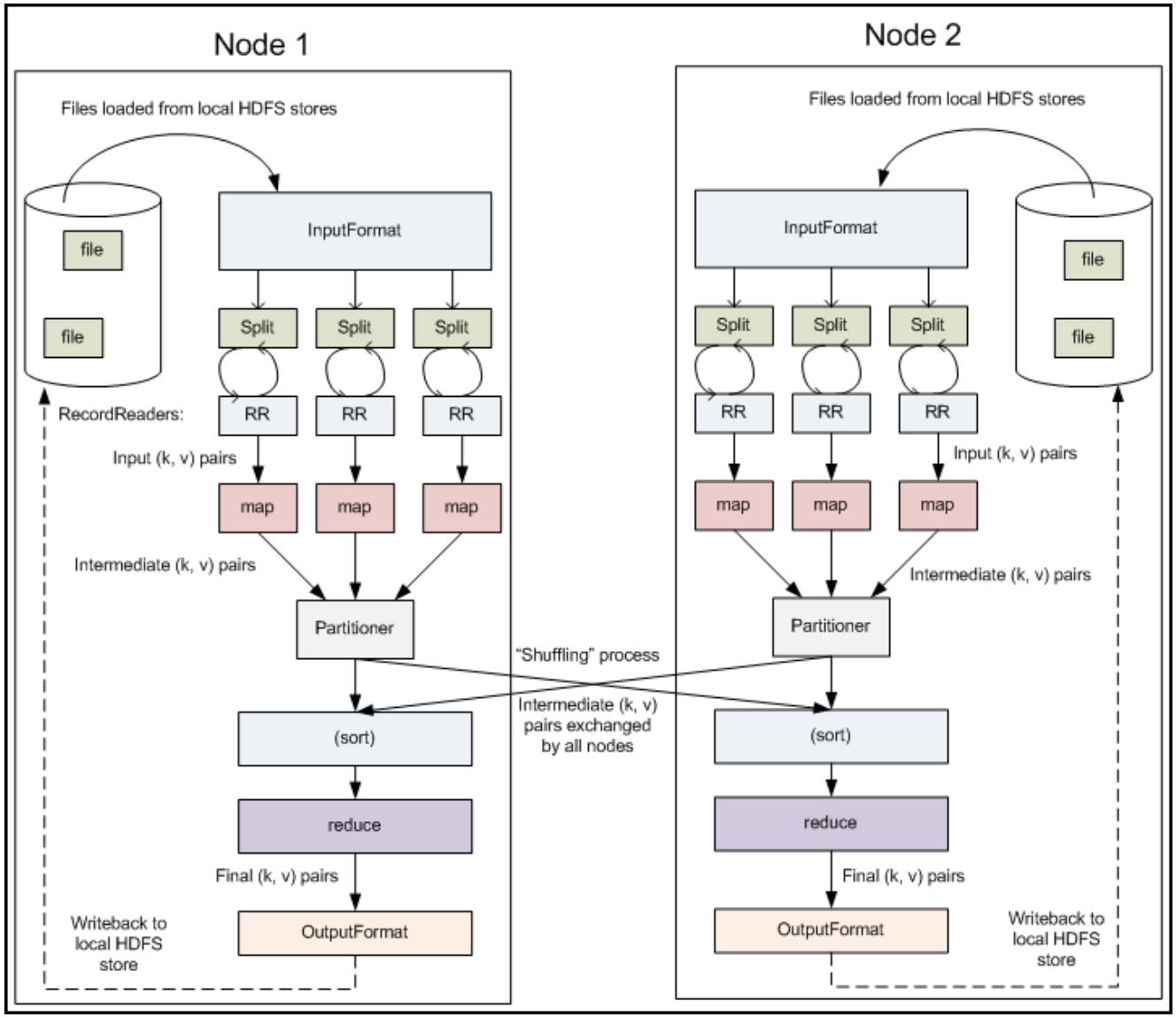


Pipeline MapReduce



Map et Reduce sont *automatiquement* réparties sur les nœuds

Pipeline MapReduce



Pipeline MapReduce

Parallélisation et distribution automatique

- HDFS se charge de la répartition et de la réplication des données ;
- Le maître divise le travail en jobs parallèles et les répartit ;
- Le maître collecte les résultats et gère les pannes des nœuds.

Là où MapReduce n'est pas bon

- *Données semi-structurées* : MapReduce est pensé pour des données indépendantes
- *Jointures* : MapReduce n'est pas prévu pour faire des jointures efficaces (à la différence d'un SGBD-R)
- *Transactions* : MapReduce n'est pas adapté à des lots de petits calculs

Pipeline MapReduce

Gestion des pannes des nœuds

Polling régulier des nœuds du cluster par le *Master*, cas de pannes :

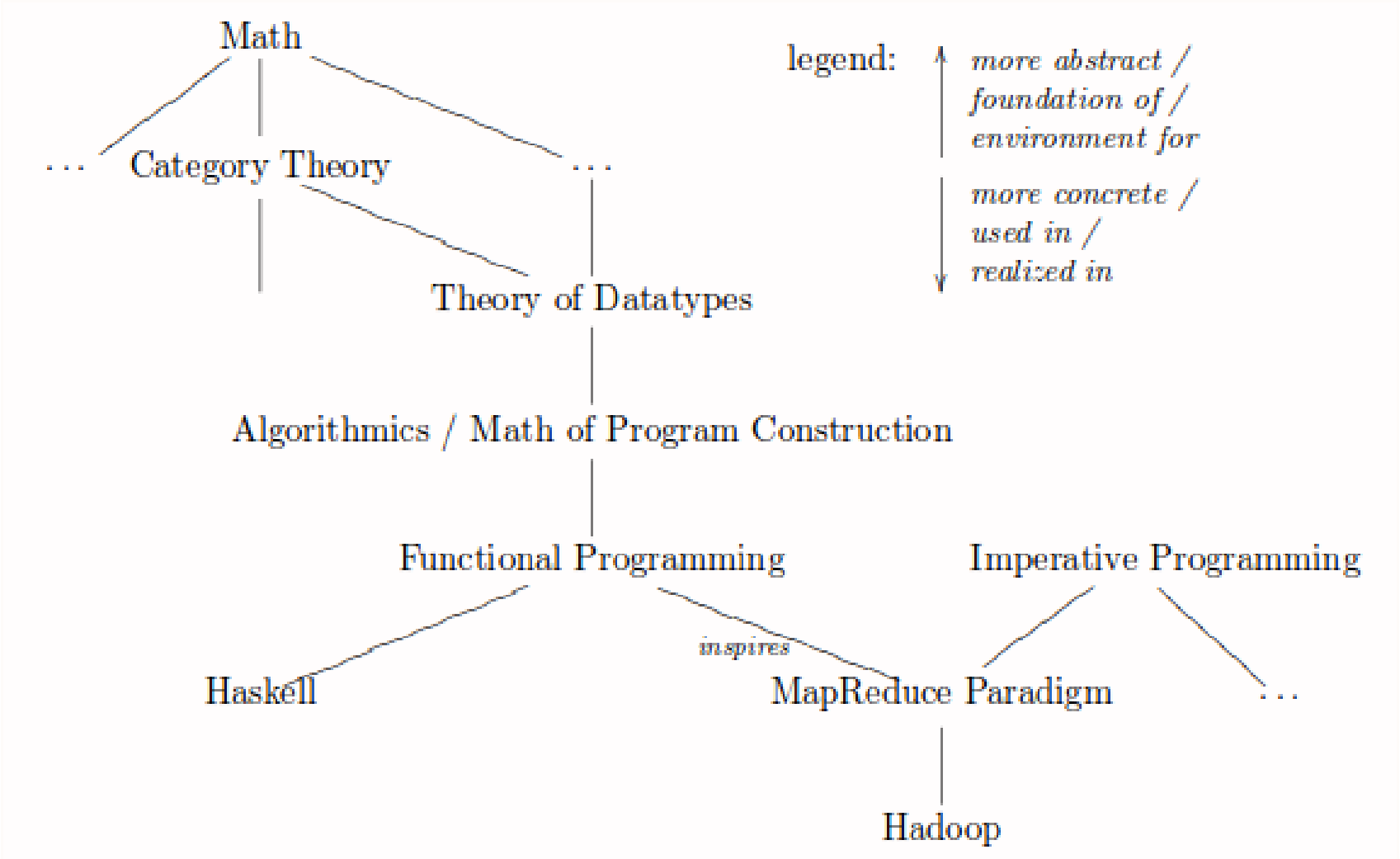
Cas du *Master* il faut tout recommencer depuis le début

Cas d'un *Reducer* le désactiver et donne les jobs en cours à d'autres

Cas d'un *Mapper* il faut :

- réattribuer ses jobs en cours
- recommencer ses jobs terminés (!)
- informer les *Reducers* du changement d'adresse.

Inspiration fonctionnelle



MapReduce, c'est (presque) de la programmation fonctionnelle

Programmation fonctionnelle

Caractéristiques

- opérations séquencées par la composition $(f \circ g)(x) = f(g(x))$:
  pas d'ordre dans les déclarations
- pas d'état en fonctionnel « pur » :
  le résultat d'une fonction ne dépend que de ses entrées
- données/variables non modifiables :
  pas d'affectation, pas de gestion explicite de la mémoire

Inspiration fonctionnelle de MapReduce

- Pipeline MapReduce, en gros : $\mathrm{reduce}(\oplus) \circ \mathrm{grp} \circ \mathrm{map}(f)$
- On peut de manière automatique paralléliser les programmes fonctionnels sur plusieurs unités de calcul

La fonction Map

$\text{map} : (A \to B) \to ([A] \to [B])$

$$\text{map}(f)[x_0, \ldots, x_n] = [f(x_0), \ldots, f(x_n)]$$

$$\text{map}(*2)[2, 3, 6] = [4, 6, 12]$$

Prototype de Map dans MapReduce

- Dans la doc. `Map : (K1, V1) -> [(K2, V2)]`
- Map est un prototype particulier du f de $\text{map}(f)$
  - On applique f sur une collection de paires clef/valeur
  - Pour chaque paire (k, v) on calcule $f(k, v)$

Exemple en pseudocode

```
function map(uri, document)
  foreach distinct term in document
    output (term, count(term, document))
```

La fonction Map

Propriétés algébriques de map

- $\mathrm{map}(id) = id$ avec $id(x) = x$
- $\mathrm{map}(f \circ g) = \mathrm{map}(f) \circ \mathrm{map}(g)$
- $\mathrm{map}(f)[x] = [f(x)]$
- $\mathrm{map}(f)(xs +\!\!+ ys) = \mathrm{map}(f)(xs) +\!\!+ \mathrm{map}(f)(ys)$

Application

- Simplification et réécriture automatique de programme
- Preuve (algébrique) d'équivalence
- Parallélisation automatique des calculs

La fonction Sort/Group/Shuffle

$\text{grp} : [(A \times B)] \to [(A \times [B])]$

$$\text{grp}[\ldots(w, a_0), \ldots, (w, a_n)\ldots] = [\ldots, (w, [a_0, \ldots, a_n]), \ldots]$$

$$\text{grp}[('a', 2), ('z', 2), ('ab', 3), ('a', 4)] = [('a', [2, 4]), ('z', [2]), ('ab', [3])]$$

Prototype de Sort/Group/Shuffle dans MapReduce

- Dans la doc. `Grp : [(K2, V2)] -> [(K2, [V2])]`
- Rappelle l'instruction `GROUP BY/ORDER BY` en SQL
- Grp est appelée de façon transparente entre Map et Reduce

La fonction Reduce

$\text{reduce} : (A \times A \to B) \to ([A] \to B)$

$$\text{reduce}(\oplus)[x_0, \ldots, x_n] = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1} \oplus x_n$$

$$\text{reduce}(+)[2, 1, 3] = 2 + 1 + 3 = 6$$

Prototype de Reduce dans MapReduce

- Dans la doc. `Reduce : (K2, [V2]) -> [(K3, V3)]`
- Reduce est un prototype particulier pour $\text{reduce}(\oplus)$
  - On applique $\oplus$ sur une collection de valeurs associées à la clef

Exemple en pseudocode

```
function reduce(term, counts)
  output (term, sum(counts))
```

La fonction Reduce

Exemples de fonctions $\mathrm{reduce}(\oplus)$

Sum : $\mathrm{reduce}(+)$

Size : $\mathrm{reduce}(+) \circ \mathrm{map}(\lambda x.1)$ où $\lambda x.1$ est la fonction constante

Flatten : $\mathrm{reduce}(+\!\!+)$ où $+\!\!+$ est la concaténation
$[a_0, \ldots, a_n] +\!\!+ [b_0, \ldots, b_m] = [a_0, \ldots, a_n, b_0, \ldots, b_m]$

Min, Max : $\mathrm{reduce}(\min)$ et $\mathrm{reduce}(\max)$ avec $\min$ et $\max$ binaires

Filter : $p\triangleleft = \mathrm{reduce}(+\!\!+) \circ \mathrm{map}(p?)$ avec
$p?(x) = [x]$ si x a la propriété p et [] sinon

Factorielle : $\mathrm{reduce}(\times)[1..n]$

La fonction Reduce

Contraintes sur $\oplus$ dans $\mathrm{reduce}(\oplus)$

- $(x \oplus y) \oplus z = x \oplus (y \oplus z)$: associatif dans le cas des listes ;
- $x \oplus y = y \oplus x$: et commutatif dans le cas des *bags* ;
- $x \oplus x = x$: et idempotent dans le cas des ensembles ;
- si $\oplus$ admet un élément neutre e, alors $\mathrm{reduce}(\oplus)[] = e$

Propriétés algébriques de reduce

- $\mathrm{reduce}(\oplus)(xs +\!\!+ ys) = \mathrm{reduce}(\oplus)(xs) \oplus \mathrm{reduce}(\oplus)(ys)$
- si $g(x \oplus y) = g(x) \otimes g(y)$ alors $g \circ \mathrm{reduce}(\oplus) = \mathrm{reduce}(\otimes) \circ \mathrm{map}(g)$
- $\mathrm{map}(f) \circ \mathrm{reduce}(+\!\!+) = \mathrm{reduce}(+\!\!+) \circ \mathrm{map}(\mathrm{map}(f))$
- $\mathrm{reduce}(\oplus) \circ \mathrm{reduce}(+\!\!+) = \mathrm{reduce}(\oplus) \circ \mathrm{map}(\mathrm{reduce}(\oplus))$
- $\mathrm{reduce}(+\!\!+) \circ \mathrm{reduce}(+\!\!+) = \mathrm{reduce}(+\!\!+) \circ \mathrm{map}(\mathrm{reduce}(+\!\!+))$

`wordcount` fonctionnel pur

Objectif

- Étant donnée une fonction $words : Doc \to [Word]$
- et une collection de documents $[Doc]$
- calculer la fonction qui compte les occurrences dans la collection $wc : [Key \times Doc] \to [Word \times \mathbb{N}]$

Étapes du calcul

$$\underset{[W\times\mathbb{N}]}{\uparrow}\ (\ \underset{W\times\mathbb{N}}{\uparrow}\ id \times (+/)\ \underset{W\times[\mathbb{N}]}{\uparrow}\ )* \cdot \underset{[W\times[\mathbb{N}]]}{\uparrow}\ grp \cdot \underset{[W\times\mathbb{N}]}{\uparrow}\ \#/ \cdot \underset{[[W\times\mathbb{N}]]}{\uparrow}\ (\ \underset{[W\times\mathbb{N}]}{\uparrow}\ (\_,1)* \cdot \underset{[W]}{\uparrow}\ words \cdot \underset{D}{\uparrow}\ exr \underset{K\times D}{\uparrow}\ )* \underset{[K\times D]}{\uparrow}$$

Où sont Map et Reduce ?

$$\overbrace{(id \times \boxed{+/})*}^{reducePerKey} \cdot \overbrace{grp}^{grpPerKey} \cdot \overbrace{\#/ \cdot (\ \boxed{(\_,1)* \cdot words \cdot exr}\ )*}^{mapPerKey}$$

wordcount en MapReduce en Java

Classe Map

```
public static class Map
  extends Mapper<Object, Text, Text, IntWritable>{
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

wordcount en MapReduce en Java

Classe Reduce

```
public static class Reduce
  extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,Conte
    throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

wordcount en MapReduce en Java

Main

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();

  Job job = new Job(conf, "wordcount");
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);
  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.waitForCompletion(true);
}
```

Autres exemples

Fréquence d'accès à partir de pages web, Map renvoie des paires `<URL,1>` et Reduce renvoie `<URL,total>`

Inversion de graphe à partir de pages web, Map renvoie des paires `<src,trg>` et Reduce renvoie `<trg,list(src)>`

Index inversé à partir de documents, Map renvoie des paires `<word,docID>` et Reduce renvoie `<word,list(docID)>`

Vecteur de termes ...

Grep distribué ...

Tri distribué ...

Langages de haut-niveau

Des langages de haut-niveau sur MapReduce

- Modèle de données rigide : clef/valeur seulement
- Pipeline fixé à deux niveaux seulement
- Code *ad hoc* nécessaire (projection, sélection)

L'aspect déclaratif de MapReduce peut être poussé plus loin, avec des langages *compilés* en MapReduce

Exemples

- Sawzall et FlumeJava par Google
- Hive/HiveQL par Facebook pour le datawarehousing
- Tenzing par Goole, implémentation de SQL
- Jaql, pour les données JSON
- Pig Latin

Pig Latin

Pig Latin

- Modèle de données riche : *relationnel imbriqué* (Pig Bag)
- Enchaînement d'opérations simples :
  - une expression transforme un Pig Bag en un autre
  - chacune est compilée en MapReduce

SQL vs PIG

SQL

```
SELECT cat, AVG(pagerank)
FROM urls
WHERE pagerank > 0.2
GROUP BY cat
HAVING COUNT(*) > 10^6
```

PIG

```
good_urls  = FILTER urls
             BY pagerank > 0.2;
groups     = GROUP good_urls BY cat;
big_groups = FILTER groups
             BY COUNT(good_urls)>10^6;
output     = FOREACH big_groups
             GENERATE cat,
               AVG(good_urls.pagerank);
```

Pig Latin

foreach	Apply one or several expression(s) to each of the input tuples
filter	Filter the input tuples with some criteria
distinct	Remove duplicates from an input
join	Join of two inputs
group	Regrouping of data
cross	Cross product of two inputs
order	Order an input
limit	Keep only a fixed number of elements
union	Union of two inputs
split	Split a relation based on a condition

Pig Latin

Exemple

```
books = load 'webdam-books.txt'
        as (year: int, title: chararray, author: chararray) ;
group_auth = group books by title;
authors = foreach group_auth
          generate group, books.author;
dump authors;
```

Résultat

```
(Foundations of Databases,
  {(Abiteboul),(Hull),(Vianu)})
(Web Data Management,
  {(Abiteboul),(Manolescu),(Rigaux),(Rousset),(Senellart)}
```

wordcount en Pig Latin

Script complet

```
input_lines = LOAD '/tmp/my-docs' AS (line:chararray);

-- Extract words from each line and put them into a bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS wo

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS

ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet'
```

MapReduce vs SGBD parallèle (1/2)

[Pavlo et al. SIGMOD09]

Hadoop MapReduce vs two parallel DBMS (one row-store DBMS and one column-store DBMS)

- Benchmark queries : a grep query, an aggregation query with a group by clause on a Web log, and a complex join of two tables with aggregation and filtering
- Once the data has been loaded, the DBMS are significantly faster, but loading is much time consuming for the DBMS
- Suggest that MapReduce is less efficient than DBMS because it performs repetitive format parsing and does not exploit pipelining and indices

MapReduce vs SGBD parallèle (2/2)

[Dean and Ghemawat, CACM10]

- Make the difference between the MapReduce model and its implementation which could be well improved, e.g. by exploiting indices

[Stonebraker et al. CACM10]

- Argues that MapReduce and parallel DBMS are complementary as MapReduce could be used to extract-transform-load data in a DBMS for more complex OLAP

MapReduce, un grand pas en arrière ?

Exégèse de l'article de DeWitt et Stonebreaker

1. MapReduce is a step backwards in database access :
   pas de schéma, de séparation physique/logique, de langage déclaratif
2. MapReduce is a poor implementation :
   pas de structures d'index (e.g., B-Tree de SGBD-R)
3. MapReduce is not novel :
   les résultats fondamentaux et techniques ont plus de 20 ans
4. MapReduce is missing features :
   pas de contraintes d'intégrité, de vues, d'updates
5. MapReduce is incompatible with the DBMS tools :
   data mining, reporting, atelier de conception

MapReduce, un grand pas en arrière ?

What is the novelty in MapReduce ?

Pas de nouveauté fondamentale, mais technique et applicative

1. évaluation de la technique du pipelining de map et reduce pour une application à de l'indexation de document
2. évaluation des performances sur cette application, avec impact du coût du transfert entre noeuds
3. montrer comment l'architecture peut être rendue *tolérante aux pannes*
4. identification des choix techniques et d'optimisations

Alternatives à Hadoop

Traitement de grandes masses de données

- Scope
- Dryad/DryadLinq
- Nephele/Pact
- Boom analytics
- Hyracks/ASTERIX

Partagent les motivations de MapReduce, en essayant de combler ses lacunes.

Références

- Patrick Valduriez : *Distributed Data Management in 2020 ?*, Keynote DEXA, Toulouse, August 30, 2011
- Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart : *Web Data Management and Distribution : Distributed Computing at Web Scale*, November 10, 2011 (*accessible gratuitement*)
- Jeffrey Dean, Sanjay Ghemawat : *MapReduce : simplified data processing on large clusters* OSDI, 2004 (*papier original*)
- Sherif Sakr, Anna Liu, Ayman G. Fayoumi : *The Family of MapReduce and Large Scale Data Processing Systems* CoRR, abs/1302.2966, 2013 (*survey sur MapReduce*)
- Maarten Fokkinga : *Background info for Map and Reduce*, 2011 (*aspects fondamentaux*)
- Ralf Lämmel : *Google's MapReduce programming model – Revisited*, Science of Computer Programming, 2008 (*programmation fonctionnelle*)