MIF04 : Gestion de données pour le Web

Framework MapReduce : fondamentaux fonctionnels

romuald.thion@univ-lyon1.fr

http://liris.cnrs.fr/ecoquery/dokuwiki/doku.php?id=enseignement:mif04

Outline

Motivation originale : Google

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

Contexte applicatif

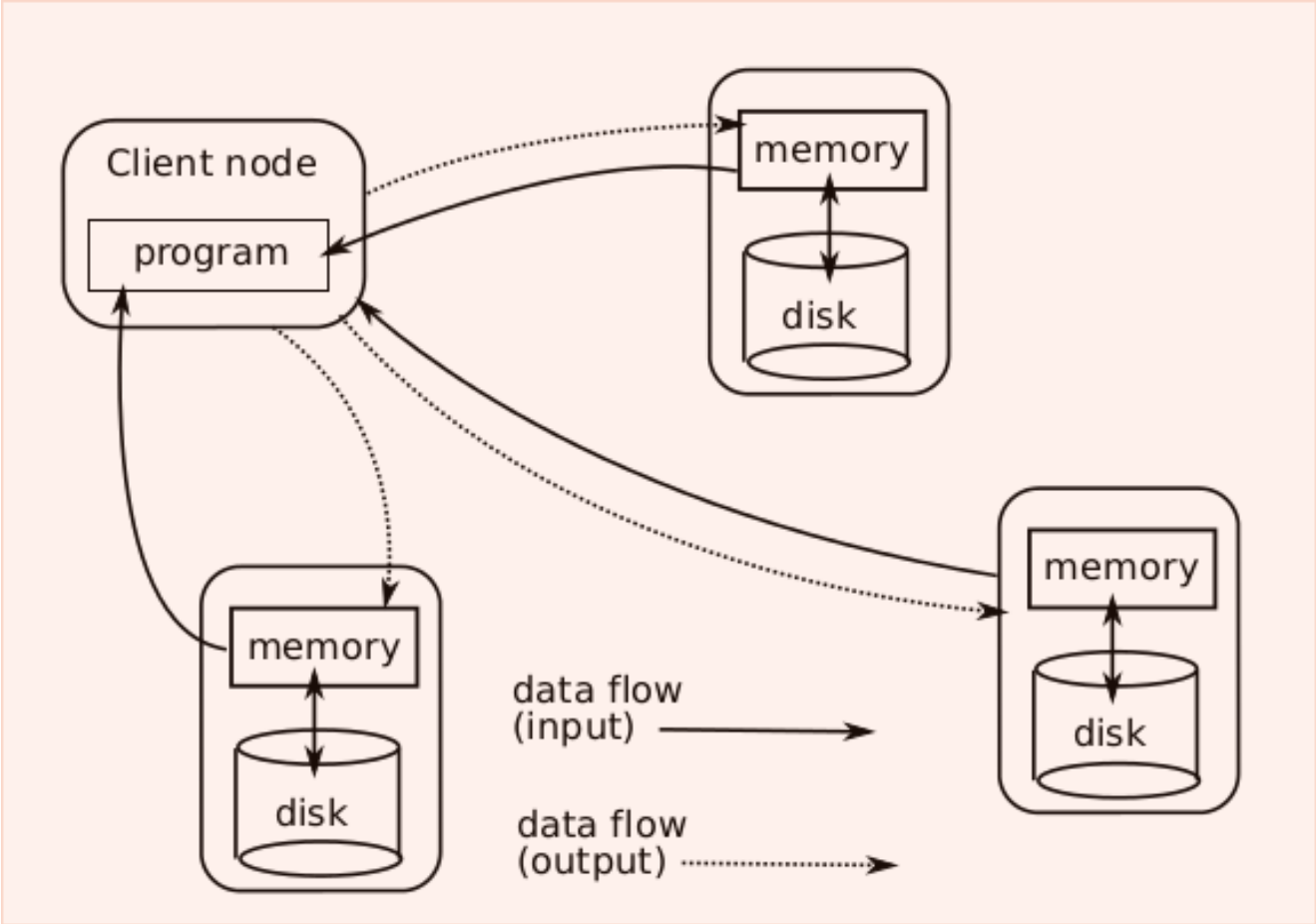
- Famille de traitement *simples* :
  index inversé, statistiques, requêtes ou mots fréquents
- Sur de *très grands* volumes de données :
  page web, log d'accès, documents

Les challenges / contraintes

Distribution des données, *parallélisation* des calculs, gestion des *pannes*, utilisable sur des *petites machines*

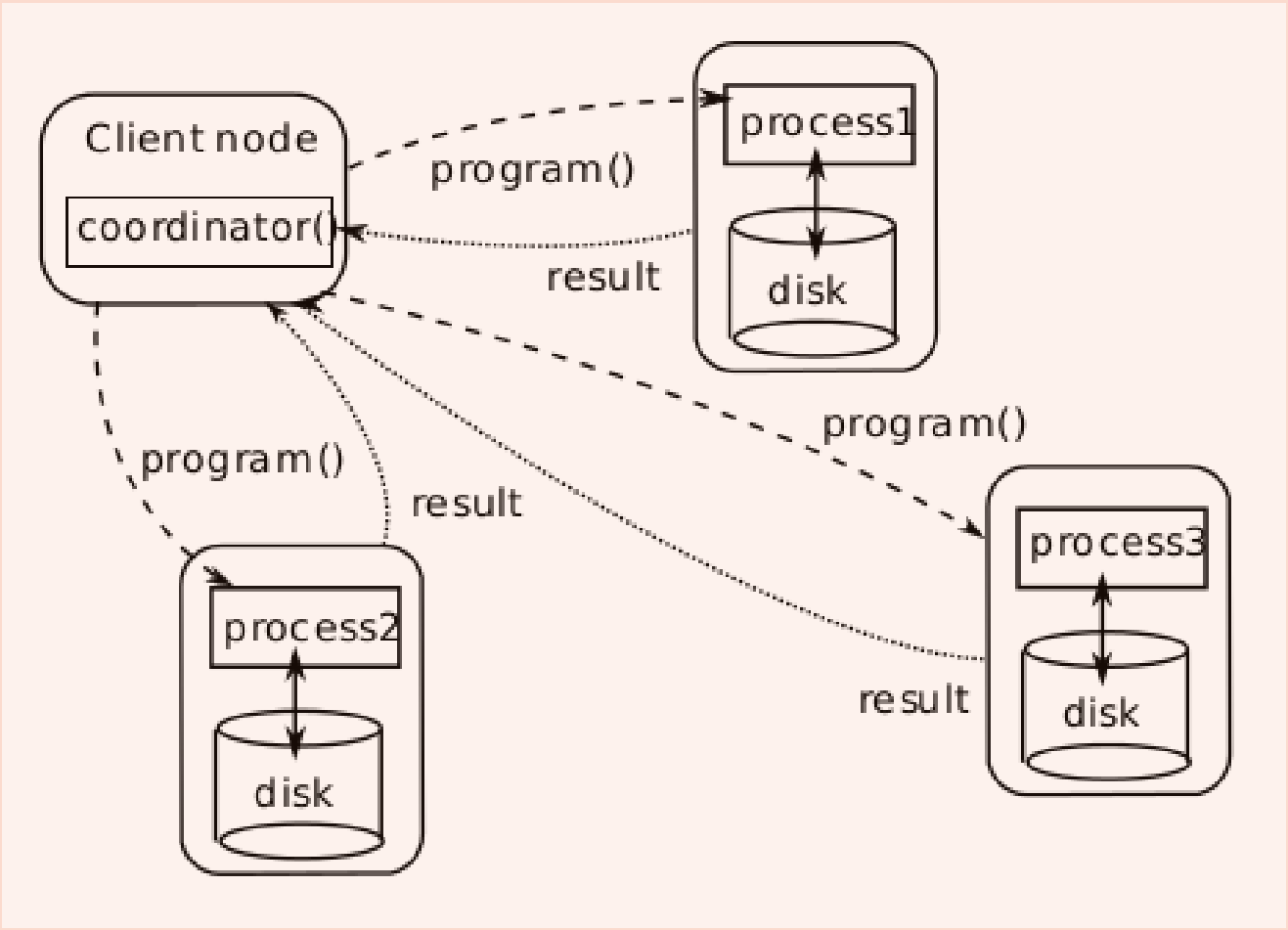
Dans cette partie, on va s'intéresser à comment et pourquoi ça marche

Distribution des données



Coût élevé des transferts réseau (*bottleneck*).

Distribution des données *et* des calculs

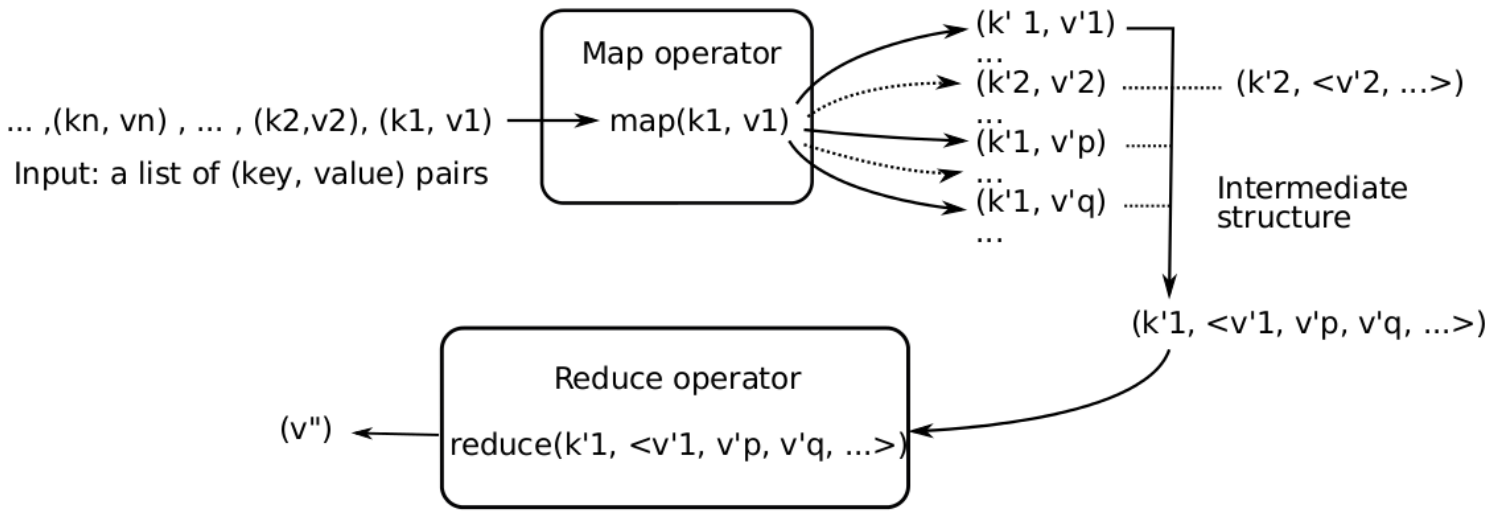


Gérer les jobs pour qu'ils s'exécutent au plus proche des données.
Principe de *data locality*.

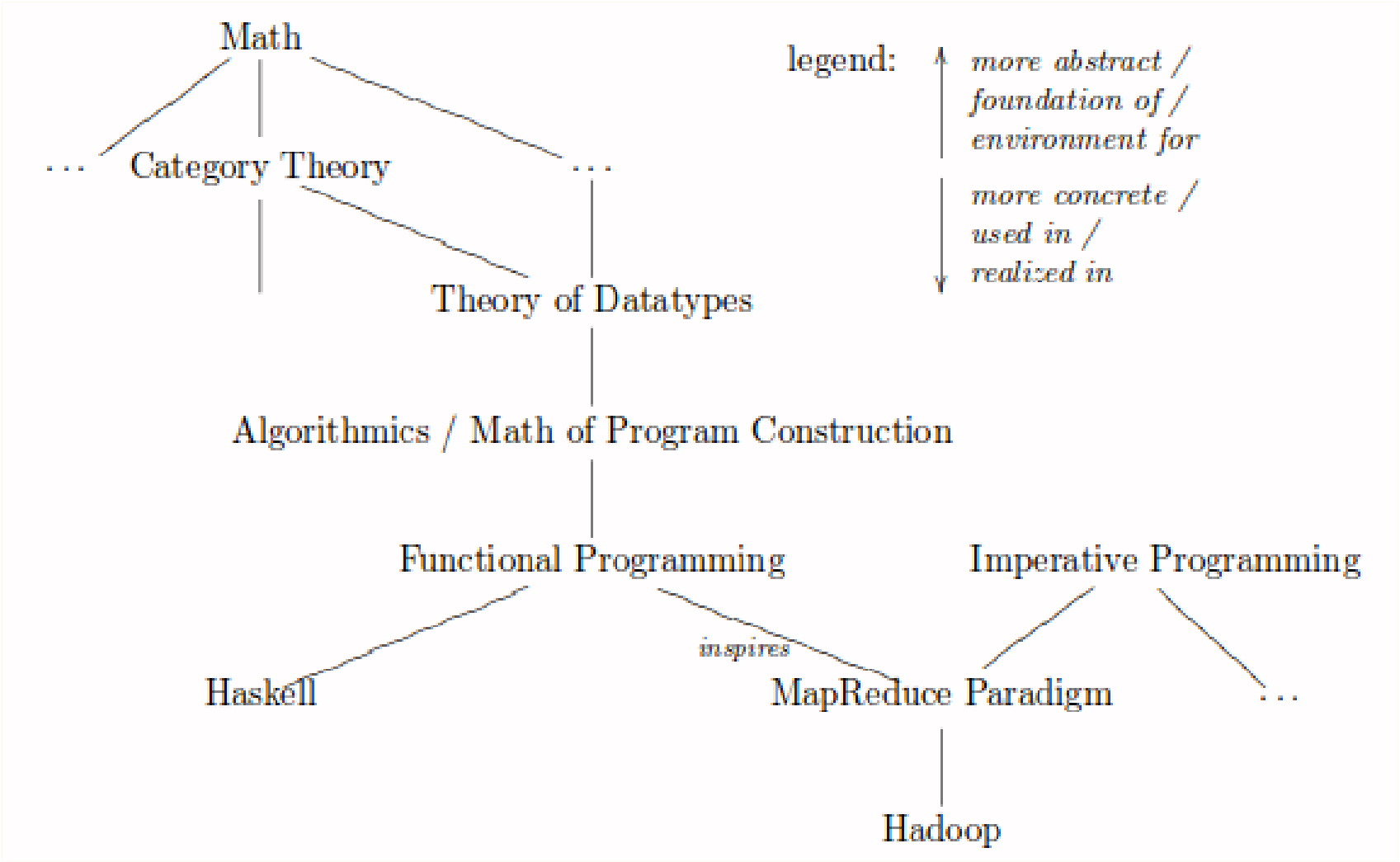
Pipeline MapReduce

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



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

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

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

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

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 $\text{reduce}(\oplus)$

- $(x \oplus y) \oplus z = x \oplus (y \oplus z)$: 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 $\text{reduce}(\oplus)[] = e$

Propriétés algébriques de reduce

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

`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

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

Où sont Map et Reduce ?

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

`wordcount` en Haskell

Haskell

- langage de programmation *fonctionnel*, *pur* et *paresseux*
- on réalise une implémentation jouet du pipeline MapReduce

```
-- from https://github.com/moizjv/haskellParallelismMapReduce

import Data.List (sortBy, groupBy) -- generice sort and group function
import Data.List.Split (splitOn)   -- tokenizer
import Data.Function (on)          -- tool

-- on :: (b -> b -> c) -> (a -> b) -> a -> a -> c
-- (.*.) `on` f = \x y -> f x .*. f y

-- sample document
egTextString :: String
egTextString = "Zola was born in Paris in 1840"

-- sample input
egInput :: [(Int, String)]
egInput = [(0, egTextString),(1, egTextString)]
```

```haskell
-- tokenizer
wordsep :: String -> [String]
wordsep = splitOn "␣"

-- add 1 to every word in a list
countOne :: [String] -> [(String, Int)]
countOne = map (\x -> (x,1))

-- mapper function : discard identifier, tokenize then count 1 for each w
mapper :: (Int,String) -> [(String,Int)]
mapper = countOne . wordsep . snd

-- sample output :
-- [ [("Zola",1),("was",1),("born",1),("in",1),("Paris",1),("in",1),("184
--   [("Zola",1),("was",1),("born",1),("in",1),("Paris",1),("in",1),("184
egOutMap :: [[(String,Int)]]
egOutMap = map mapper egInput

-- sort according to tokens
combining :: [(String,Int)] -> [(String,Int)]
combining =  sortBy (compare `on` fst)
```

```
-- once sorted, group values of identical tokens
-- NB : \l -> ( fst . head $ l , map snd l ) :: [(a, b)] -> (a, [b])
-- keep the head a-value and group all b-values.
grp :: [(String,Int)] -> [(String,[Int])]
grp = map (\l -> (fst . head $ l, map snd l)) . groupBy ((==) `on` fst)

-- sample output :
-- [("1840",[1,1]),("Paris",[1,1]),("Zola",[1,1]),("born",[1,1]),("in",[1,
egOutGrp :: [(String,[Int])]
egOutGrp = grp . combining . concat $ egOutMap

-- reducer function : simply add lists of 1s using foldr function
red :: (String,[Int]) -> (String, Int)
red (s,ls) = (s, foldr1 (+) ls)

-- definition of foldr :
---- if the list is empty, the result is the initial value z; else
---- apply f to the first element and the result of folding the rest
--foldr f z []     = z
--foldr f z (x:xs) = f x (foldr f z xs)
```

```haskell
-- complete mapreduce pipeline
mapreduce :: [(Int, String)] -> [(String, Int)]
mapreduce = map red . (grp . combining) . concat . map mapper

-- complete mapreduce pipeline with local aggregation
mapreduce' :: [(Int, String)] -> [(String, Int)]
mapreduce' = map red . (grp . combining) . concat . map (mapper') where
               mapper' :: (Int,String) -> [(String,Int)]
               mapper' = map red . grp . combining . mapper

-- sample output
-- [("1840",2),("Paris",2),("Zola",2),("born",2),("in",4),("was",2)]
egOutMR :: [(String, Int)]
egOutMR = mapreduce egInput

-- generic map reduce pipeline
simpleMapReduce
     :: ((k1,v1)    -> [(k2,v2)])        -- fonction 'mapper'
     -> ([(k2,v2)] -> [(k2,[v2])])       -- fonction 'sort/group/shuffle'
     -> ((k2,[v2]) -> (k3,v3))           -- fonction 'reduce'
     -> [(k1,v1)]                        -- input : key/values
     -> [(k3,v3)]                        -- output
simpleMapReduce m c r = map r . c . concat . map m
```

Autres applications

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 ...

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, mais apportent des réponses différentes.

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*)