MIF04 – Gestion de données pour le Web
TD – Map/Reduce en MongoDB

Algorithm 1 Word count (algorithmique)

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
5:         end for
6:     end method
7: end class
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:         end for
7:         Emit(term t, count sum)
8:     end method
9: end class
```

Algorithm 2 Word count `javascript` pour MongoDB

```
var map = function() {
  var content =  this.content.toLowerCase().split(/[\s,!-.;?_:*]+/);
  // tokenization using js String.split() with a regex
  // \s matches a single white space character, including space, tab, form feed
      , line feed.
  // see https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/
      Regular_Expressions
  for (var i = 0, n = content.length; i < n; i++) {
    if (content[i])
      emit(content[i], 1);
  }
};

var reduce = function(key, values) {
  var sum = 0;
  for (var i = 0, n = values.length; i < n; i++) {
    sum = sum + values[i];
  }
  return sum;
};
```

Exercice 1 : analyse et extension du problème *word count* (*[2, Chapitre 3]*)

L'algorithme 1 définit en pseudo-code les classes MAPPER et REDUCER d'HADOOP pour le calcul du nombre d'occurrences de mots dans une collection de documents. L'algorithme 2 donne une implémentation équivalente en `javascript` pour MONGODB. Pour l'exemple, on considère la collection de paires clef/valeur suivante :

Doc1 « *appreciate the fold* »

Doc2 « *cata equals fold* »

Doc3 « *fold the fold* »

1. En supposant que les documents **Doc1** et **Doc2** se situent sur un premier nœud et **Doc3** sur un autre, donner les résultats intermédiaires obtenus par chaque MAPPER.
2. En supposant que l'espace des mots soit divisé en deux, les mots inférieurs à « *f* » sur un nœud et ceux après « *f* » sur un autre, donner les résultats intermédiaires obtenus des REDUCER qui regroupent les clefs puis les résultats finaux des REDUCER. Discuter la pertinence de cette règle de répartition des clefs dans ce cas d'application.
3. On considère maintenant une agrégation locale des données : chaque MAPPER va calculer le nombre d'occurrence de chaque mot dans sa collection de documents. Donner le pseudo-code ou le code `javascript` en utilisant un *tableau associatif* (appelé aussi *dictionnaire*) pour stocker les calculs intermédiaires.
4. Reprendre la première question avec cette nouvelle version.
5. La classe MAPPER d'Hadoop dispose également des méthodes :
   — INITIALIZE qui est appelée *avant* le parcours de la collection
   — CLOSE qui est appelée *après* le parcours de la collection
   Proposez une variante du MAPPER avec tableau associatif qui utilise ces deux méthodes pour effectuer un maximum d'agrégations dans le MAPPER.
6. Reprendre la première question avec cette nouvelle version.
7. On souhaite construire un index inversé, qui à chaque mot associe la *liste des identifiants de documents où le mot apparaît*. Donner le code `javascript` pour construire l'index inversé.

Exercice 2 : calcul de moyenne (*[2, Chapitre 3]*)

1. Donner le pseudo-code ou le code `javascript` qui calcule la moyenne des valeurs associées à une clef, e.g., $\text{REDUCE}(w, [1, 2, 3, 4]) = (w, 2.5)$ dans le cas où le `reduce` est appelé *exactement une fois sur chaque clef*, comme c'est le cas dans Hadoop.
2. Proposer une variante qui fonctionne en MONGODB où il y a des *re-reduce*. Il faudra utiliser la fonction `finalize` de MONGODB qui a le même prototype que `reduce` mais est appelée une fois tous les REDUCER terminés.

Exercice 3 : algèbre relationnelle en Map/Reduce (*[3, Chapitre 2.3]*)

Dans cet exercice on s'intéresse à implémenter les principaux opérateurs de SQL avec des tâches Map/Reduce. Pour les exemples, on considère deux relations $R(A, B)$ et $S(X, Y)$. On considère qu'un tuple $R(a, b)$ est représenté dans MONGODB par un objet `json` de la forme $\{\_id : i, value : \{rel : R, A : a, B : b\}\}$ où i est un identifiant automatiquement généré par MONGODB. Donner les codes `javascript` correspondant aux requêtes suivantes. Pour la recherche dans un tableau `json`, on utilisera la fonction `Array.indexOf(x)` qui renvoie le premier index où `x` est dans `Array` et -1 sinon.

1. ```
   SELECT *
   FROM R
   WHERE A < 1;
   ```
   ```

2. ```
   SELECT DISTINCT A,B
   FROM R
   ```

3. ```
   SELECT *
   FROM R
   INTERSECT
   SELECT X as A, Y as B
   FROM S
   ```

4. ```
   SELECT *
   FROM R
   UNION
   SELECT X as A, Y as B
   FROM S
   ```

5. ```
   SELECT A, count(*)
   FROM R
   GROUP BY A
   ```

6. ```
   SELECT R.A, R.B, S.Y
   FROM R INNER JOIN S on R.A = S.X
   ```

7. On aimerait calculer une jointure sur des collections différentes en MongoDB, malheureusement, ce n'est pas possible. Proposer une solution alternative.

Exercice 4 : définitions fonctionnelles des fonctions Map et Reduce (*[1]*)

On s'intéresse à un cas particulier de tâches Map/Reduce qui sont représentables par la composition suivante, où $g : (K_1 \times V_1) \to [(K_2 \times V_2)]$ et $(\oplus) : V_2 \times V_2 \to V_2$ avec grp : $[(K_2 \times V_2)] \to [(K_2 \times [V_2])]$ la fonction de regroupement. Le pipeline complet est définissable ainsi :

$$\text{mapreduce}(g)(\oplus) = \text{map}((x, ys) \mapsto (x, \text{red}(\oplus)(ys))) \circ \text{grp} \circ (+\!\!+) \circ \text{map}(g)$$

On donne les définitions inductives de $\text{red}(\oplus)$ et de $\text{map}(g)$ sur des listes non-vides :

$$\begin{aligned}
\text{red}(\oplus)[x] &= x \\
\text{red}(\oplus)(xs +\!\!+ ys) &= (\text{red}(\oplus)(xs)) \oplus (\text{red}(\oplus)(ys)) \\
\text{map}(g)[x] &= [g(x)] \\
\text{map}(g)(xs +\!\!+ ys) &= \text{map}(g)(xs) +\!\!+ \text{map}(g)(ys)
\end{aligned}$$

1. En utilisant le même formalisme, donner les types des fonctions $(+\!\!+)$, map et red. Vérifier que la définition de $\text{mapreduce}(g)(\oplus)$ est correctement typée et que son type est $[(K_1 \times V_1)] \to [(K_2 \times V_2)]$.
2. En utilisant uniquement des définitions fonctionnelles, donner les définitions de g et $\oplus$ pour que $\text{mapreduce}(g)(\oplus)$ calcule :
   — le nombre de valeurs associées à chaque clef ;
   — le nombre d'occurrence de chaque valeur ;
   — le nombre total de valeurs ;
   — le nombre total de valeurs *différentes* ;
3. Montrer que la loi $\oplus$ doit être associative, on utilisera pour cela la définition de $\text{red}(\oplus)$ et l'associativité de $+\!\!+$.
4. À partir de la preuve précédente pour les listes, donner l'argument pour prouver que $\oplus$ doit être commutatif dans le cas des multi-ensembles et idempotent dans le cas des ensembles.
5. Montrer que si $g(x \oplus y) = g(x) \otimes g(y)$ alors $g \circ \text{red}(\oplus) = \text{red}(\otimes) \circ \text{map}(g)$ par induction sur les listes.
   ```

Références

[1] R. Lämmel. Google's mapreduce programming model – revisited. *Science of Computer Programming*, 70(1) :1 – 30, 2008.

[2] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010. http://lintool.github.io/MapReduceAlgorithms/.

[3] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, Cambridge, 2012.