

Les requêtes de type *Map Reduce*

Romain Tavenard

1 Avant-propos

Dans la suite de ce TD, vous allez formuler des requêtes de type *Map Reduce* pour des bases MongoDB. Ces requêtes vous permettront d'accéder à des fonctionnalités non offertes nativement par MongoDB, comme :

- grouper par clé et non par valeur ;
- compter les occurrences de mots dans les chaînes de caractères de la base ;
- effectuer des opérations arithmétiques non autorisées par MongoDB (nous verrons ici le cas de la division d'un entier par une durée) ;
- *etc.*

Ces requêtes *Map Reduce* ont en revanche un inconvénient principal : elles sont exécutées en *Javascript* et sont donc relativement lentes (cela aura toutefois peu d'effet sur des bases de petite taille).

1.1 Éléments de syntaxe *Javascript*

Une requête *Map Reduce* est de la forme suivante :

```
db.nomDeLaCollection.mapReduce(fonction_map,
                                fonction_reduce,
                                "nomDeLaCollectionDeSortie")
```

et le résultat est enregistré dans la table "nomDeLaCollectionDeSortie".

Il faut donc être capable de coder en *Javascript* deux fonctions (**map** et **reduce**). La première ne prend pas d'argument, ne retourne rien, mais *émet* des messages par des appels à la fonction **emit(clé, valeur)**. La seconde prend deux arguments en entrée :

- une clé émise par un ou plusieurs appels à **map**;
- une liste de valeurs émises par les mêmes appels à **map**.

Pour coder ces fonctions, il est nécessaire de maîtriser quelques concepts de base en *Javascript*.

Tout d'abord, pour définir une variable en *Javascript*, la syntaxe à utiliser est la suivante :

```
var nomDeLaVariable = valeur;
```

Ensuite, pour appeler une fonction (par exemple la fonction `emit`), la syntaxe est relativement usuelle :

```
emit(clé, valeur);
```

Enfin, pour parcourir une liste (ou un dictionnaire), la syntaxe sera :

```
for (var i in liste) {  
    valeur = liste[i];  
}  
  
for (var clé in dictionnaire) {  
    valeur = dictionnaire[clé];  
}
```

1.2 Un exemple de requête *Map Reduce*

Supposons tout d'abord que l'on souhaite faire la somme des valeurs associées à l'attribut `"a"` pour l'ensemble des documents d'une collection nommée `collec` (notez qu'une telle requête aurait pu être effectuée plus efficacement sous la forme d'une requête d'agrégation).

La formulation *Map Reduce* de cette requête sera donc de la forme :

```
function map_somme() {  
    emit("a", this["a"]);  
}  
  
function reduce_somme(key, values) {  
    s = 0;  
    for (var i in values) {  
        s = s + values[i];  
    }  
    return s;  
}  
  
db.collec.mapReduce(map_somme, reduce_somme, "output_somme")  
db.output_somme.find()
```

Vous remarquerez que :

- dans la fonction `map_somme`, l'accès au document courant se fait par le biais de la variable `this` ;
- la fonction `reduce_somme` a une valeur de retour qui est du même type que la valeur émise par la fonction `map_somme`.

2 À vous de jouer

1. Créez une collection contenant les documents suivants :

```
{a: 123, b: 5}  
{b: 4, c: 8}  
{c: 4}
```

```
db.collec.insert([{"a": 123, b: 5}, {"b": 4, c: 8}, {"c": 4}])
```

2. Affichez, pour chaque clé, le nombre de document dans lesquels cette clé est présente.

```
function map_compte() {  
  for(var key in this) {  
    emit(key, {count: 1});  
  }  
}  
function reduce_compte(key, emits) {  
  total = 0;  
  for(var i in emits) {  
    total += emits[i]["count"];  
  }  
  return {count: total};  
}  
  
db.collec.mapReduce(map_compte, reduce_compte, "output_collec")  
db.output_collec.find()
```

3. Affichez les mots présents dans la collection issue du fichier `random_text.json` ainsi que leur fréquence d'apparition. Classez ces résultats par ordre de fréquence décroissante.

```
function map_words() {  
  words = this["text"].split(" ");  
  for (var i in words) {  
    emit(words[i], {count: 1});  
  }  
}  
function reduce_words(key, emits) {  
  total_counts = 0;  
  for (var i in emits) {  
    total_counts += emits[i]["count"];  
  }  
  return {count: total_counts};  
}
```

```
db.random_text.mapReduce(map_words, reduce_words, "output_words")
db.output_words.find().sort({"value.count": -1})
```

4. La collection issue du fichier `links.json` contient une liste de liens. À chaque lien est associé un certain nombre de catégories (attribut `tags`). On aimerait classer les différentes catégories à l'aide du score S_c suivant :

$$S_c = \sum_{d \in \mathcal{D}_c} \frac{s_d}{\Delta_d}$$

où \mathcal{D}_c est l'ensemble des documents contenant la catégorie c , s_d est la valeur de l'attribut `score` du document d et Δ_d est l'ancienneté du document d , calculée comme :

```
anciennete = new Date() - date_du_document
```

Proposez une requête qui permette de classer les liens selon ce critère.

```
function map_tags() {
  for (var i in this["tags"]) {
    score = this["score"] / (new Date() - this.date);
    emit(this["tags"][i], {score: score});
  }
}
function reduce_tags(key, emits) {
  total_score = 0;
  for (var i in emits) {
    total_score += emits[i]["score"];
  }
  return {score: total_score};
}

db.links.mapReduce(map_tags, reduce_tags, "output_tags")
```

3 Pour aller plus loin

5. Supposons maintenant que nous ayons, dans une collection, des documents constitués de deux attributs : `"x"` et `"y"` (cf. `reg.json`). Proposez un moyen, dans le formalisme *Map Reduce*, de calculer le coefficient de régression linéaire $\hat{\beta}$ optimal au sens des moindres carrés. Vérifiez votre résultat en proposant une autre solution utilisant les opérateurs d'agrégation.

NB: vous pourrez utiliser pour cela l'argument `finalize` des `options` de `mapReduce` qui prend pour valeur une fonction `f` de la forme :

```

function f(cle, document) {
    [...]
    return document_modifié;
}

function map_reg() {
    var xx = this.x * this.x;
    var xy = this.x * this.y;
    emit(0, {"xx": xx, "xy": xy});
}

function reduce_reg(key, emits) {
    s_xx = 0.;
    s_xy = 0.;
    for (var i in emits) {
        s_xx += emits[i]["xx"];
        s_xy += emits[i]["xy"];
    }
    return {"xx": s_xx, "xy": s_xy};
}

function finalize_reg(key, doc_final) {
    return {"beta": doc_final.xy / doc_final.xx};
}

db.reg.mapReduce(map_reg, reduce_reg, {"out": "output_reg", "finalize": finalize_reg})

```