Tutoriel pour implémenter un algorithme de MapReduce en JavaScript classique et ES6

Par Marc AUTRAN

Date de publication : 16 août 2016

CONFIRMÉ

Durée : 2 heures

Cet article montre comment utiliser JavaScript et surtout sa dernière mouture (ES6) pour implémenter un algorithme simple de MapReduce. L'objectif de ce tutoriel est de disséquer l'algorithme en l'implémentant de bout en bout sans recourir à une bibliothèque spécialisée.

I - Introduction....3
I-A - Les prérequis....3
I-B - L'algorithme MapReduce....3
I-C - Le cahier des charges....3
I-C-1 - La fonction Map....4
I-C-2 - les fonctions Sort et Shuffle....4
I-C-3 - La fonction Reduce....4
II - Les solutions qui existent dans l'écosystème JavaScript....4
II-A - Les modules Node.js....4
II-B - Implémentation en JavaScript classique (ES5)....5
II-B-1 - Implémentation en EcmaScript 6....5
III - Conclusions et remerciements....6

I - Introduction

I-A - Les prérequis

Ce tutoriel s'adresse à des lecteurs qui connaissent déjà le langage JavaScript.

Nous n'utiliserons pour ce tutoriel que le langage JavaScript implémenté en standard dans les navigateur (EcmaScript 5) ainsi que sa dernière mouture (EcmaScript 6) également en standard dans le framework Node.js (JavaScript coté serveur).

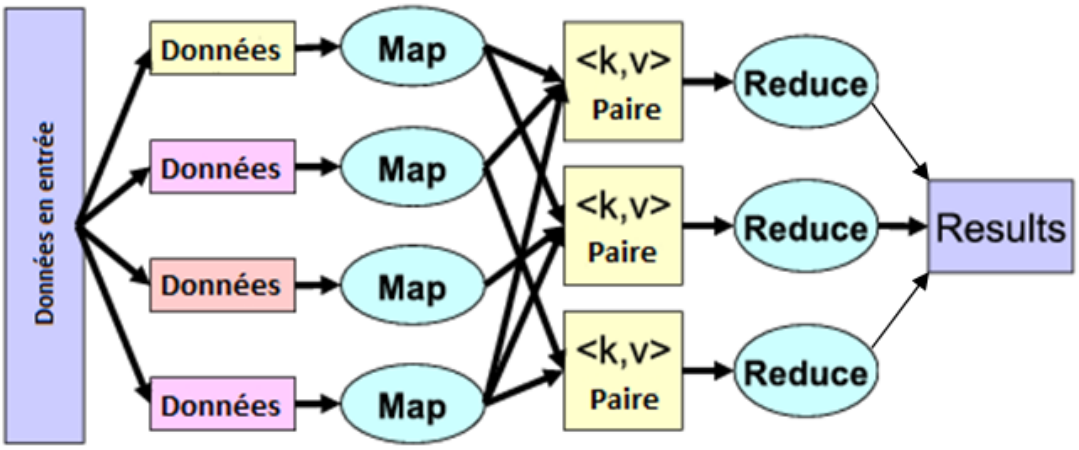
Aucune librairie complémentaire ne sera nécessaire dans ce tutoriel.

I-B - L'algorithme MapReduce

Ce modèle de programmation popularisé par Google permet, à travers 2 macro-fonctions, de découper un problème en sous-problèmes (phase de Map), puis de collecter les résultats issus de la résolutions de ces sous problèmes pour fournir un résultat global (phase de Reduce).

Dans un conexte de Big Data, les sous-problèmes issus de la phase de Map sont distribués sur différents nœuds pour être traités en parrallele.

L'essentiel des traitements consiste à créer des listes de clés/valeurs à partir de données brutes et de les triés.



Dans le cadre de cet article, l'exemple pédagogique que nous traiterons ne comprendra qu'un sous problème traiter dans son ensemble par un nœud unique.

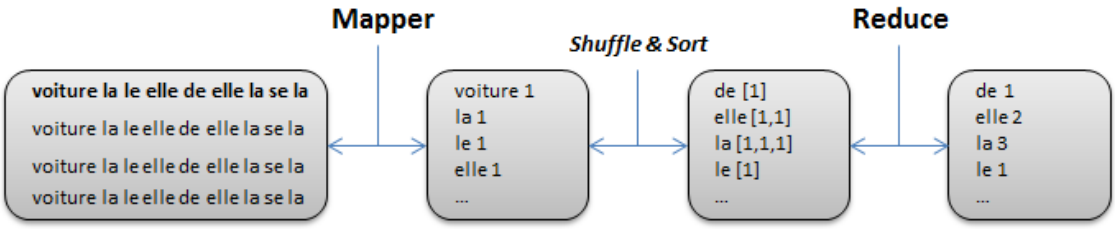
I-C - Le cahier des charges

Dans son **Tutoriel Hadoop**, Mickael Baron décrit très clairement le patron de conception MapReduce et nous en propose une implémentation en langage Java sur un exemple pédagogique très simple. L'objet de cet article, est de réaliser sur le même exemple un traitement similaire en utilisant JavaScript.

Pour expliquer les concepts de map et de reduce, nous prendrons donc l'exemple du compteur de mots fréquemment utilisés, avec une légère variante. Tous les mots sont comptabilisés à l'exception du mot « se ». Le jeu de données à analyser sera le suivant :

```
var inputString = "voiture la le elle de elle la se la maison voiture";
```

La figure ci-dessous énumère les différentes étapes qui seront présentées par la suite.



I-C-1 - La fonction Map

Cette première étape consistera à retournée la liste de clés/valeurs suivante :

[['voiture', [1]], ['la', [1]], ['le', [1]], ['elle', [1]], ['de', [1]], ['elle', [1]], ['la', [1]], ['se', [0]], ['la', [1]], ['maison', [1]], ['voiture', [1]]]

On remarque que dans cette exemple la liste des clés/valeurs est constituée en ajoutant à cette liste le mot rencontré (la clé) et sa valeur qui sera suivant notre règle de gestion de 1 sauf pour la clé « se » qui doit naturellement valoir 0.

I-C-2 - les fonctions Sort et Shuffle

Ces 2 fonctions doivent nous permmettrent de constituer une liste dans laquelle chaque clés est associée à la liste de toutes ses valeurs. Ce qui doit nous donner dans notre exemple :

[['de', [1]], ['elle', [1, 1]], ['la', [1, 1, 1]], ['le', [1]], ['maison', [1]], ['voiture', [1, 1]]]

on remarque que la clé « se » à disparue conformément aux directives initiales.

I-C-3 - La fonction Reduce

Cette dernière étape doit présenter un résultat synthétique des opérations précédentes.

Dans le cas de notre compteur de mots, il suffit de sommer pour chaque clé les valeurs de sa liste associée :

[['de', 1], ['elle', 2], ['la', 3], ['le', 1], ['maison', 1], ['voiture', 2]]

II - Les solutions qui existent dans l'écosystème JavaScript

II-A - Les modules Node.js

Sans surprise, on découvre qu'il existe quelques modules node.js pour réaliser rapidement ce travail. J'utilise personnellement dans mes activités de Big Data le module **npm MapReduce**qui permet aisément d'implémenter l'algorithme de la façon suivante :

```
var mr = new maprereduce(function(item) {
    // la fonction de map
}, function(result) {
    // la fonction de reduce
}, function(item1, item2) {
    // la fonction de reduce finale globale
});
```

Pour installer ce plugin on fait classiquement : `npm install maprereduce`.Mais pour rester dans le JavaScript pédagogique et vraiment comprendre le MapReduce, nous n'utiliserons pas cette bibliothèque. En effet l'objectif de cet article est de comprendre dans les détails les mécanismes de cet algorithme en l'implémentant en pur JavaScript.

II-B - Implémentation en JavaScript classique (ES5)

Le JavaScript se prête particulièrement à ce type de traitement car ce langage offre un objet Array qui possède un arsenal de méthodes pour les tris, les extractions ou les recherches ...

Nous utiliserons les méthode map() et reduce() de l'objet Array. Et pour la partie *shuffle et sort* de l'algorithme, on s'appuiera sur les methodes sort() et filter() de ce même objet.

Voici le code source de l'algorithme qui fonctionne sur tout les navigateurs.

```
var inputString = "voiture la le elle de elle la se la maison voiture";
var inputArray = inputString.split(" ");
//étape du map qui retourne une liste clé/valeur
var mappedArray = inputArray.map(function(str){
  if (str.valueOf() === "se")
    return [str, [0]];
  else
    return [str, [1]];
});
//elimine 'se'
var shuffledArray = mappedArray.filter(function(array){
  return array[1][0] > 0;
});
//tri sur la clé
var sortedArray = shuffledArray.sort();
//etape du shuffle qui regroupe des valeur par clé
var size = sortedArray.length - 1;
for (var i = 0; i < size; i++){
  if (sortedArray[i][0] == sortedArray[i+1][0]){
    sortedArray[i][1].push(1);
    sortedArray.splice(i+1, 1);
    size--;
    i--;
  }
}
//étape du reduce qui renvoie un tableau de clé-->nombre d'occurences
for (var i = 0; i < sortedArray.length; i++)
{
  sortedArray[i][1] = sortedArray[i][1].reduce(function(a, b) {
    return a + b;
  });
}
console.log(sortedArray);
```

II-B-1 - Implémentation en EcmaScript 6

Il n'était pas possible de présenter l'implémentation de cet algorithme sans proposer une déclinaison en ES6 pour utiliser la toute nouvelle notation fonctionnelle.

Les développeurs Node.js utilisent particulièrement cette notation fonctionnelle et ES6 en général. Or c'est bien coté serveur que le Big Data est majoritairement implémenté.

Voici le code commenté en ES6 :

```
var inputString = "voiture la le elle de elle la se la maison voiture";
var inputArray = inputString.split(" ");
//étape du map qui retourne une liste clé/valeur
var mappedArray = inputArray.map(str => (str.valueOf() === "se") ? [str, [0]] : [str, [1]]);
```

```
//elimine 'se'
var shuffledArray = mappedArray.filter(elem => elem[1][0] > 0);
//tri sur la clé
var sortedArray = shuffledArray.sort();
//etape du shuffle qui regroupe des valeurs par clé
sortedArray.forEach((elem, index, arr) => {
    while (arr[index + 1] && elem[0] === arr[index + 1][0]){
        arr[index][1].push(1);
        arr.splice(index + 1, 1);
    }
});
//étape du reduce qui renvoie un tableau de clé-->nombre d'occurences
var finalArray = sortedArray.map(elem => [elem[0],elem[1].reduce((a, b) => a + b)]);
console.log(finalArray);
```

III - Conclusions et remerciements

Nous avons vu comment implémenter un algorithme de MapReduce en JavaScript. Et nous avons pu constater que ce langage particulièrement bien gréé pour la manipulation des tableaux offrait une grande perméabilité aux algorithmes de types MapReduce. Cependant, cet article ne doit rester qu'une introduction à ce type d'algorithme et n'a nullement l'ambition de poser les bases du Big Data. En effet le Big Data est traité dans l'écosystème Javscript en liaison avec Hadoop via des modules Node.js. Cette symbiose fera l'objet de tutoriels encore au stade de la rédaction.

Nous tenons à remercier XXX et XXX pour la relecture technique et XXX pour la relecture orthographique de cet article.