Bases de données documentaires et distribuées, http://b3d.bdpedia.fr

Frameworks MapReduce: MongoDB

Rappels : Notions et vocabulaire

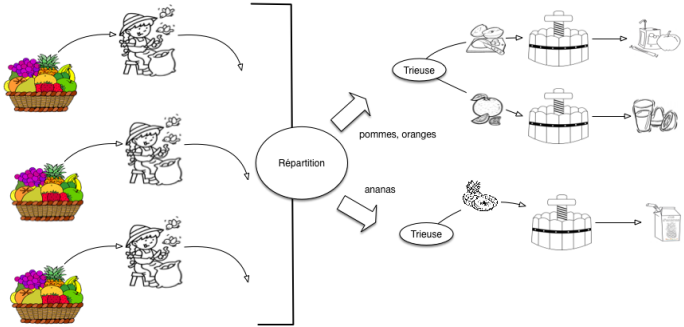
Vous devriez savoir :

- **Item (entrée)**, **document** en ce qui nous concerne.
- **Fonction de Map**, implante la transformation appliquée à un item pendant la phase de Map.
- **Paire intermédiaire**, c'est le résultat de la fonction de Map.
  Une paire intermédiaire (k, v) comprend l'identifiant (étiquette) du groupe auquel v appartient.
- **Groupe intermédiaire**, ensemble des valeurs produites par la fonction de Map et partageant le même identifiant de groupe.
- **Fonction de Reduce**, implante la transformation appliquée un groupe pendant la phase de Reduce.

Ne pas retenir (pour l'instant) : comment tout cela s'exécute en distribué, avec parallélisation et reprise sur panne.

La notion de *framework*

Framework = environnement assumant les aspects **génériques** d'un traitement.
Ici : tout ce qui n'est pas le cuisinier, le pressoir .



- Impose un modèle d'exécution contraint.
- Mais il suffit de fournir la spécification **fonctionnelle** : $F_{map}()$ et $F_{red}()$

Exemple : compter les fruits sains, par catégorie

$F_{map}()$ qui teste si un produit est pourri et **émet** une paire intermédiaire.

```
function controleFruit (fruit)
{
   if (fruit.statut != pourri) {
     emit (fruit.type, 1)
   }
}
```

Chaque groupe intermédiaire contient autant de 1 que de fruits sains.

$F_{red}()$ additionne les valeurs dans un groupe.

```
function compterPomme (typeFruit, groupe)
{
  return <typeFruit, sum(groupe)>
}
```

La fonction est appliquée à **chaque** groupe.

Il reste à soumettre ces deux fonctions au *framework*. **Clair ?**

MapReduce et MongoDB

MongoDB comprend un moteur de calcul MapReduce.

Les fonctions doivent être écrites en Javascript.

Dans ce qui suit : deux exemples de traitements MapReduce pour

- Comprendre les principes, le raisonnement.
- Réaliser les limites quand il s'agit d'obtenir des traitements un peu complexes.
- Méditer sur l'utilité de la chose...

Fonctions fournies : à tester avec l'utilitaire `mongo` ou avec RoboMongo (ou tout autre client).

Exemple : regroupement de films

On veut regrouper les films par réalisateur

La fonction F_{map}.

```
var mapRealisateur = function() {
                  emit(this.director._id, this.title);
          };
```

La fonction F_{red}.

```
   var reduceRealisateur = function(directorId, titres) {
      var res = new Object();
      res.director = directorId;
      res.films = titres;
      return res;
    };
```

Soumission au *framework*

```
db.movies.mapReduce(mapRealisateur,  reduceRealisateur, {out: {"inline": 1}} )
```

Le résultat

Chaque exécution de F_{red} produit un document de la forme.

```
{
        "_id" : "artist:3",
        "value" : {
                "director" : "artist:3",
                "films" : [
                        "Vertigo",
                        "Psychose",
                        "Les oiseaux",
                        "Pas de printemps pour Marnie",
                        "La mort aux trousses"
                ]
        }
}
```

On aimerait connaître le réalisateur : exercice !

Options d'exécution : voir le support.

Jointure : la méthode

Notre base est celle contenant les films **avec références**.

Films et artistes sont dans des documents **distincts**. Le but est d'obtenir pour chaque artiste la liste des films qu'il/elle a réalisé.

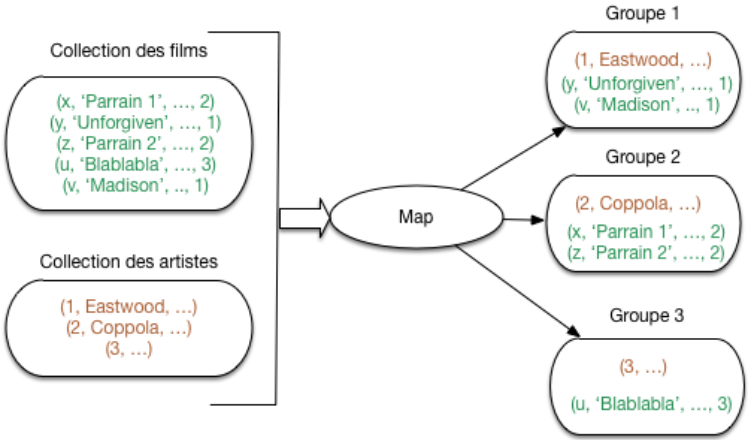
Principe (général) : on exploite le mécanisme de regroupement pour associer, dans un même groupe les informations à combiner.

On va créer autant de groupes que d'artiste. Dans chaque groupe on place :

- l'artiste dont l'identifiant correspond à l'identifiant du groupe ;
- les films (0, 1 ou plusieurs) dont l'identifiant **du metteur en scène** correspond à l'identifiant du groupe.

Méthode très représentative de l'application de MapReduce à des algorithmes complexes (pas trop quand même).

Illustration : groupement d'un metteur en scène et de ses films



En pratique : la fonction de Map

```
var mapJoin = function() {
   // Est-ce que l'id du document contient le mot "artist"?
   if (this._id.indexOf("artist") != -1) {
     // Oui ! C'est un artiste. Ajoutons-lui son type.
     this.type="artist";
     // On produit une paire avec pour id celle de l'artiste
     emit(this._id, this);
   }
   else {
     // Non: c'est un film. Ajoutons-lui son type.
    this.type="film";
    // Simplifions un peu le document pour l'affichage
    delete this.summary;
    delete this.actors;
     // On produit une paire avec pour id celle du metteur en sc.
    emit(this.director._id, this);
  }
 };
```

En pratique : la fonction de Reduce

```
var reduceJoin = function(id, items) {

  var director = null, films={result: []}

  // On cherche l'artiste dans cette liste
  for (var idx = 0; idx < items.length; idx++) {
    if (items[idx].type=="artist") {
         director = items[idx];
     }
  }

  // Maintenant, 'director' contient l'artiste : on l'affecte aux films
  for (var idx = 0; idx < items.length; idx++) {
     if (items[idx].type=="film"  && director != null) {
         items[idx].director = director;
         films.result.push (items[idx]);
      }
   }
   return films;
};
```