Certificat Data Scientist

MongoDB

Nicolas Klutchnikoff

Octobre 2021

MongoDB, une base de données orientée documents

Présentation

Il s'agit d'un **logiciel libre** sous double licences (GNU AGPL v3.0 et Apache v2.0).

La communcation se fait selon le principe **client-serveur**. Il s'agit d'un système de gestion de base de données populaire et réputé facile d'utilisation :http://db-engines.com/en/ranking

La langue maternelle de MongoDB est le **JavaScript** pour l'exécution de fonctions côté serveur. Les objets manipulés sont au format **BSON** (JSON binaire).

NoSQL

MongoDB fait partie de la mouvance NoSQL au titre de système de gestion de base de données **orienté documents**.

Les données manipulées sont des documents enrichis d'un champ `_id` et enregistrés dans des **collections**.

- **Imbrication de documents** : cela permet une **approche relationnelle**.
- **Absence de schéma** : pour créer un document (ou une collection), il suffit de l'utiliser mais cette simplicité implique que **toute faute de frappe peut avoir des conséquences importantes**.

Package `mongolite`

Le package `mongolite` permet de se connecter à un serveur MongoDB local ou distant et d'interagir avec lui depuis R. Son installation demande que les bibliothèques de développement de OpenSSL et Cyrus SASL soient présentes sur le système mais tout est expliqué à l'installation.

```
install.packages("mongolite")
```

Nous pouvons ensuite charger le package pour commencer à travailler avec MongoDB depuis R.

```
library(mongolite)
```

Connexion

La première étape est de se connecter au serveur MongoDB. Par défaut, la connexion est locale (**localhost**) mais elle peut être distante.

```
user <- "nicolasK"
pass <- "ofadNa7mSmFRCWLS"
host <- "cluster0.umn7x.mongodb.net"
path <- "CEPE"
opts <- "retryWrites=true&w=majority"
url <- str_c("mongodb+srv://", user, ":", pass, "@", host, "/", path, "?", opts)
```

Pour ouvrir la connexion avec le serveur MongoDB, nous utilisons la fonction **mongo** avec un nom de collection pour récupérer un objet **R** permettant d'interagir avec elle.

```
collection <- "change_me" # Choisir un nom pour votre collection
m <- mongo(collection) # connectrion locale ! mongo(collection, url = url) sinon
```

Premier contact

La collection n'a pas besoin d'être préalablement créée et l'objet retourné par `mongo` offre plusieurs méthodes pour la manipuler. Par exemple, `count` sans paramètre retourne le nombre de documents.

```
m$count()
```

```
## [1] 0
```

Si la collection a déjà été utilisée, elle peut être vidée avec la méthode `drop`.

```
if(m$count() > 0) m$drop()
```

Attention, sur un serveur partagé, la collection peut être en cours d'utilisation par une autre personne ! Il est conseillé de choisir un nom original pour faire des essais sans être perturbé.

Insertion

La méthode `insert` permet d'ajouter des éléments à la collection.

```
# Cartes du jeu "The Lord of the Rings: The Card Game"
m$insert(fromJSON("https://ringsdb.com/api/public/cards"))
```

```
## List of 5
##  $ nInserted  : num 1100
##  $ nMatched   : num 0
##  $ nRemoved   : num 0
##  $ nUpserted  : num 0
##  $ writeErrors: list()
```

```
m$count()
```

```
## [1] 1100
```

Insertion

L'absence de schéma permet d'insérer des documents qui n'ont pas la même structure.

```
m$insert(list(name="Luke Skywalker", outlier=TRUE))
```

```
## List of 6
##  $ nInserted  : int 1
##  $ nMatched   : int 0
##  $ nModified  : int 0
##  $ nRemoved   : int 0
##  $ nUpserted  : int 0
##  $ writeErrors: list()
```

```
m$count()
```

```
## [1] 1101
```

Recherche

La méthode `find` permet de faire une recherche dans la collection. Sans paramètre, toute la collection est retournée.

```
m$find()
```

Il est possible de définir des critères de recherche avec l'argument `query`. Le format JSON est utilisé pour cela.

```
m$find(query='{"type_name": "Contract"}')
```

L'argument `fields` permet de limiter les champs retournés (par défaut, `_id` est toujours retourné).

```
m$find(query='{"type_name": "Contract"}', fields='{"name": 1, "illustrator": 1}')
```

Recherche - Exemple

```
m$find(query='{"type_name": "Contract"}',
       fields='{"_id": 0, "pack_name": 1, "name": 1, "illustrator": 1}')
```

```
##                          pack_name                      name     illustrator
## 1              A Shadow in the East                 Fellowship  Leanna Crossan
## 2                   Wrath and Ruin       The Burglar's Turn  Greg Bobrowski
## 3               The City of Ulfast Forth, The Three Hunters!   Justin Gerard
## 4       Challenge of the Wainriders        The Grey Wanderer   Justin Gerard
## 5           Under the Ash Mountains      Council of the Wise   Borja Pindado
## 6                The Land of Sorrow    Messenger of the King   Justin Gerard
## 7             The Fortress of Nurn       Bond of Friendship   Borja Pindado
## 8           ALeP - Children of Eorl        The Last Alliance  Unknown Artist
## 9  ALeP - The Scouring of the Shire            Into the West Donato Giancola
## 10     The Hunt for the Dreadnaught        A Perilous Voyage            <NA>
```

Recherche - Opérateurs

Il est possible d'utiliser des opérateurs dans l'argument `query`. Ils sont précédées du caractère '$'.

- `$exists` pour tester l'existence d'un champ,

```
m$find(query='{"outlier": {"$exists": true} }', fields='{"_id": 0, "name": 1}')
```

```
##              name
## 1 Luke Skywalker
```

- `$and`, `$or`, `$nor` et `$not` pour la logique,

```
m$find(query='{"$and": [{"type_name": "Contract"}, {"illustrator": "Leanna Crossan"
       fields='{"_id": 0, "name": 1}')
```

```
##          name
## 1 Fellowship
```

Recherche - Opérateurs

- `$regex` pour utiliser des expressions régulières,

```
m$find(query='{"name": {"$regex": "^Z", "$options" : "i"} }',
       fields='{"_id": 0, "name": 1}')
```

```
##          name
## 1 Zigil Miner
```

- `$lt`, `$lte`, `$gt` et `$gte` pour comparer des nombres,

```
m$find(query='{"attack": {"$gte": 4} }',
       fields='{"_id": 0, "name": 1, "attack": 1}') %>% head(3)
```

```
##        name attack
## 1   Gandalf      4
## 2   Saruman      5
## 3 Treebeard      4
```

Recherche - Opérateurs

- `$ne` pour tester la non-égalité,

```
m$find(query='{"pack_name": {"$ne": "Core Set"} }',
       fields='{"_id": 0, "pack_name": 1, "name": 1}') %>% head(3)
```

```
##              pack_name           name
## 1 The Hunt for Gollum  Bilbo Baggins
## 2 The Hunt for Gollum  Dúnedain Mark
## 3 The Hunt for Gollum Campfire Tales
```

- Et bien d'autres : `$in` pour l'inclusion, `$nin` pour la non-inclusion, `$all` pour l'inclusion stricte, `$size` pour la taille d'un tableau, `$type` pour le type d'une valeur, `$mod` pour le modulo, ...https://docs.mongodb.com/manual/reference/operator/query/

Compter

La méthode `count` permet de compter les documents qui vérifient des conditions données.

```
m$count('{"pack_name": "Core Set"}')
```

```
## [1] 73
```

```
m$count('{"$and": [
  {"pack_name": "Core Set"},
  {"type_name": {"$in": ["Hero", "Contract"]}}
]}')
```

```
## [1] 12
```

Valeurs uniques

La méthode **`distinct`** retourne la liste des valeurs prises par une clé parmi les documents vérifiant une recherche donnée.

```
m$distinct(key="type_name")
```

```
## [1] "Ally"              "Attachment"        "Contract"
## [4] "Event"             "Hero"              "Player Side Quest"
## [7] "Treasure"
```

```
m$distinct(key="illustrator", query='{"pack_name": "The Hunt for Gollum"}')
```

```
##  [1] "Ben Zweifel"       "David A. Nash"     "Felicia Cano"
##  [4] "Jake Murray"       "John Gravato"      "Joko Mulyono"
##  [7] "Katherine Dinger"  "Magali Villeneuve" "Stu Barnes"
## [10] "Tony Foti"
```

Tri

L'argument `sort` de la méthode `find` permet de trier les résultats selon les valeurs d'un champ (`1` pour l'ordre croissant et `-1` pour le décroissant).

```
m$find(
  query='{"threat": {"$exists": true} }',      # Supprime les NA
  fields='{"_id": 0, "name": 1, "threat": 1}', # Le nom et le niveau de menace
  sort='{"threat": -1}'                        # En partant du plus menaçant
) %>% head(5)
```

```
##        name threat
## 1   Gandalf     14
## 2    Elrond     13
## 3 Treebeard     13
## 4   Saruman     13
## 5   Gwaihir     13
```

Suppression

Nous avons déjà vu `drop` pour vider une collection. La méthode `remove` supprime les entrées correspondant à une recherche donnée.

```
m$count()
```

```
## [1] 1101
```

```
m$remove(query='{"outlier": {"$exists": true} }')
m$count()
```

```
## [1] 1100
```

Itération

La méthode `iterate` prend les mêmes arguments que `find` et retourne un **itérateur** pour parcourir le résultat de la recherche ligne par ligne à l'aide de **$one** (ou par page avec **$page**). Une fois le dernier élément retourné, l'itérateur devient obsolète.

```
it <- m$iterate(query='{"$and": [{ "threat": {"$gte": 13} },
                                 { "health": {"$gte":  5} }]}',
                fields='{"_id": 0, "name": 1}',
                sort='{"name": 1}')
while(!is.null(item <- it$one())) {
  cat("Respect à toi, ô", item$name, "!\n")
}
```

```
## Respect à toi, ô (MotK) Beorn !
## Respect à toi, ô Gandalf !
## Respect à toi, ô Treebeard !
```

Import/Export

Le format par défaut pour ces opérations est le NDJSON (Newline Delimited JSON) que nous avons évoqué pour la gestion des flux avec **jsonlite**.

Les méthodes **export** et **import** sont très simples.

```
# Exporte la collection vers un fichier
m$export(file("/tmp/dump.json"))
# Purge la collection
m$drop(); m$count()
```

```
## [1] 0
```

```
# Importe la collection depuis un fichier
m$import(file("/tmp/dump.json"))
m$count()
```

```
## [1] 1100
```

Import/Export - Gestion des flux

Grâce aux fonctions `stream_in` et `stream_out` de `jsonlite`, nous pouvons mettre en place des mécanismes élégants pour importer des données obtenues par flux (réseau, capteur, ...).

Un flux se décompose en **pages** de taille donnée et le principe est de stocker chaque page (fichier temporaire, mémoire, ...) avant de l'importer dans la collection.

L'avantage de cette approche est que nous pouvons faire agir une fonction `handler` à ce moment pour **manipuler les données avant l'import**.

Import/Export - Gestion des flux (Exemple)

La variable `traits` des données de The Lord of the Rings : The Card Game est une chaîne de caractères qui contient plusieurs informations. Ce n'est pas bien formaté et nous souhaitons la remplacer par un tableau contenant ces informations pour tester l'inclusion (opérateur `$in`).

```
m$find(fields='{"_id": 0, "name": 1, "traits": 1}') %>% head(2)
```

```
##       name                    traits
## 1  Aragorn Dúnedain. Noble. Ranger.
## 2 Théodred   Noble. Rohan. Warrior.
```

```
m$find(query='{"traits": {"$in": ["Dwarf"]} }',
       fields='{"_id": 0, "name": 1, "traits": 1}')
```

```
## data frame with 0 columns and 0 rows
```

Import/Export - Gestion des flux (Exemple)

```
# Purge la collection
m$drop()
# Fichier temporaire pour stocker les pages
ftmp <- file(tempfile(), open="w+b")
# Importation d'un flux depuis un fichier (400 documents par page)
file("/tmp/dump.json") %>%
  stream_in(pagesize=400,          # Taille des pages
            handler=function(df) { # Handler de page
              cat(" [", nrow(df), "ligne(s) lue(s) ]")
              df$traits <- df$traits %>%
                strsplit(split=' ') %>%
                sapply(function(s) { gsub('\\.', '', s) })
              stream_out(df, ftmp)
              m$import(ftmp)
            })
# Fermeture du fichier temporaire
close(ftmp)
```

Import/Export - Gestion des flux (Exemple)

```
## using a custom handler function.

## opening file input connection.

##  [ 400 ligne(s) lue(s) ]Complete! Processed total of 400 rows.
##  Found 400 records... [ 400 ligne(s) lue(s) ]Complete! Processed total of 400 ro
##  Found 800 records... [ 300 ligne(s) lue(s) ]Complete! Processed total of 300 ro
##  Found 1100 records...

## closing file input connection.
```

Import/Export - Gestion des flux (Exemple)

Il est possible de rendre la sortie moins verbeuse avec l'option `verbose=FALSE`.

Voici le résultat :

```
m$find(fields='{"_id": 0, "name": 1, "traits": 1}') %>% head(2)
```

```
##       name                   traits
## 1  Aragorn Dúnedain, Noble, Ranger
## 2 Théodred   Noble, Rohan, Warrior
```

```
m$find(query='{"traits": {"$in": ["Dwarf"]} }',
       fields='{"_id": 0, "name": 1, "traits": 1}') %>% head(2)
```

```
##    name                traits
## 1 Glóin           Dwarf, Noble
## 2 Gimli Dwarf, Noble, Warrior
```

Modification

Pour modifier un document de la collection, la méthode `update` procède en deux temps :

- rechercher le(s) document(s) à modifier,
- apporter les modifications au(x) document(s).

Par défaut, un seul document peut être modifier. Pour appliquer les modifications à plusieurs documents, il faut utiliser l'argument `multiple=TRUE`.

La recherche se fait avec l'argument `query` et les modifications avec `update`.

Modification - Changer une valeur

Utilisons `$set` pour changer la valeur d'une clé (ou créer la paire clé/valeur).

```
m$update(query='{"name": "Gandalf"}',
         update='{"$set": {"cheveux": "gris"}}')
```

```
m$find(query='{"name": "Gandalf"}', fields='{"_id": 0, "name": 1, "cheveux": 1}')
```

```
##      name cheveux
## 1 Gandalf    gris
## 2 Gandalf    <NA>
## 3 Gandalf    <NA>
## 4 Gandalf    <NA>
## 5 Gandalf    <NA>
```

Modification - Changer une valeur

Utilisons `$set` pour changer la valeur d'une clé (ou créer la paire clé/valeur).

```
m$update(query='{"name": "Gandalf"}',
         update='{"$set": {"cheveux": "gris"}}',
         multiple=TRUE) # Tous les documents sont modifiés

m$find(query='{"name": "Gandalf"}', fields='{"_id": 0, "name": 1, "cheveux": 1}')
```

```
##      name cheveux
## 1 Gandalf    gris
## 2 Gandalf    gris
## 3 Gandalf    gris
## 4 Gandalf    gris
## 5 Gandalf    gris
```

Modification - Supprimer une paire clé/valeur

La suppression d'une paire se fait avec `$unset`.

```
m$update(query='{"name": "Gandalf"}',
         update='{"$unset": {"cheveux": "gris"}}',
         multiple=TRUE) # Tous les documents sont modifiés
```

```
m$find(query='{"name": "Gandalf"}', fields='{"_id": 0, "name": 1, "cheveux": 1}')
```

```
##      name
## 1 Gandalf
## 2 Gandalf
## 3 Gandalf
## 4 Gandalf
## 5 Gandalf
```

Modification - Autres

De nombreuses commandes sont disponibles :

- `$push`, `$pull`, `$addToSet`, ... pour modifier un tableau,
- `$inc`, `$mul`, ... pour modifier une valeur numérique,
- `$rename` pour renommer une clé,
- ...

https://docs.mongodb.com/manual/reference/operator/update/

À vous de jouer !

Agrégation et MapReduce

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par Sphere et par Type.

Méthode brutale

```
v_sphere <- m$distinct(key="sphere_name")
v_type <- m$distinct(key="type_name")
tdc <- matrix(0L, nrow=length(v_sphere), ncol=length(v_type),
              dimnames=list(v_sphere, v_type))
for(sphere in v_sphere) {
  for(type in v_type) {
    query <- paste0('{"$and": [{"sphere_name":"', sphere, '"},',
                               '{"type_name":"', type, '"}]}')
    tdc[sphere, type] <- m$count(query)
  }
}
print(tdc)
```

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par Sphere et par Type.

Méthode brutale

```
##            Ally Attachment Contract Event Hero Player Side Quest Treasure
## Baggins       0          0        0     2    2                0        0
## Fellowship    0          3        0     3    8                0        0
## Leadership   73         50        0    65   52                2        0
## Lore         77         49        0    62   53                2        0
## Neutral      22         34       10    22    1                2       42
## Spirit       71         47        0    60   52                2        0
## Tactics      67         50        0    62   51                2        0
```

- C'est très moche !
- Il y a beaucoup d'aller-retours entre le client et le serveur.

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par Sphere et par Type.

Méthode R

```
table(m$find(fields='{"_id": 0, "sphere_name": 1, "type_name": 1}'))
```

```
##                    sphere_name
## type_name           Baggins Fellowship Leadership Lore Neutral Spirit Tactics
##   Ally                    0          0         73   77      22     71      67
##   Attachment              0          3         50   49      34     47      50
##   Contract                0          0          0    0      10      0       0
##   Event                   2          3         65   62      22     60      62
##   Hero                    2          8         52   53       1     52      51
##   Player Side Quest       0          0          2    2       2      2       2
##   Treasure                0          0          0    0      42      0       0
```

Motivation - Table de contingence

Nous souhaitons compter le nombre de cartes par Sphere et par Type.

Méthode R

```
table(m$find(fields='{"_id": 0, "sphere_name": 1, "type_name": 1}'))
```

- C'est plus propre !
- Une seule requête mais elle retourne un objet potentiellement volumineux.
- Tous les calculs se font côté client.

Agrégateurs

Un agrégateur est une fonction qui regroupe les valeurs contenues dans plusieurs documents sélectionnés et retourne une structure contenant des objets “simples” et “plus informatifs”.

Agrégateurs

Les méthodes `count` et `distinct` sont des agrégateurs.

```
m$count('{"type_name": "Hero"}')
```

Agrégateurs

Les méthodes `count` et `distinct` sont des agrégateurs.

```
m$distinct(key="pack_name", query='{"type_name": "Hero"}')
```

Pipeline d'agrégation

Le **pipeline d'agrégation** de MongoDB est une séquence de **stages** à traverser pour transformer des documents en un résultat agrégé.

Même si nous en verrons un autre dans la suite, il s'agit du modèle d'agrégation **privilégié** de MongoDB.

Les stages filtrent, transforment, groupent, trient, ... les documents dans un ordre établi. Par exemple :

- `$match` (filtre comme avec `query`),
- `$group` (regroupe et accumule),
- `$sort` (trie).

Pour les autres stages (`$project`, `$limit`, `$skip`, `$sample`, `$out`,...), voir :https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

Pipeline d'agrégation - Méthode `aggregate`

La méthode `aggregate` permet de contruire un agrégateur basé sur le modèle du pipeline d'agrégation. Cette méthode prend un tableau en paramètre contenant les différentes étapes à réaliser.

Pipeline d'agrégation - Méthode `aggregate`

La méthode `aggregate` permet de contruire un agrégateur basé sur le modèle du pipeline d'agrégation. Cette méthode prend un tableau en paramètre contenant les différentes étapes à réaliser.

Illustration avec `count` :

```
m$aggregate('[
  { "$match": { "type_name": "Hero" } },
  { "$group": { "_id": null, "count": { "$sum": 1 } } }
]')
```

```
##   _id count
## 1  NA   219
```

Pipeline d'agrégation - Méthode `aggregate`

```
m$aggregate('[
  { "$match": { "type_name": "Hero" } },
  { "$group": { "_id": null, "count": { "$sum": 1 } } }
]')
```

Quelques explications :

- `$match` est similaire à `query`,
- le champ `_id` de `$group` reçoit la clé utilisée pour les groupes ou `null` pour considérer tous les documents,
- le champ `count` de `$group` est le nom de l'**accumulateur** défini en suivant,
- la valeur des accumulateurs est maintenue groupe par groupe,
- la définition des accumulateurs est évaluée pour chaque document.

Pipeline d'agrégation - Exemple de `distinct`

Pour imiter la méthode `distinct`, nous pouvons utiliser

```
m$aggregate('[
  { "$group": { "_id": "$sphere_name" } }
]')
```

```
##          _id
## 1     Spirit
## 2    Neutral
## 3    Baggins
## 4 Fellowship
## 5       Lore
## 6    Tactics
## 7 Leadership
```

Pipeline d'agrégation - Exemple de `distinct`

Pour imiter la méthode `distinct`, nous pouvons utiliser

```
m$aggregate('[
  { "$group": { "_id": "$sphere_name" } }
]')
```

ou bien

```
m$aggregate('[
  { "$group": { "_id": null, "sphere_name": { "$addToSet": "$sphere_name" } } }
]')
```

```
##   _id                                                    sphere_name
## 1  NA Leadership, Spirit, Neutral, Baggins, Fellowship, Lore, Tactics
```

Pipeline d’agrégation - Tri

Voici comment compter des effectifs et les trier :

```
m$aggregate('[
  { "$group": { "_id": "$sphere_name", "count": { "$sum": 1 } } },
  { "$sort": { "count": -1 } }
]')
```

```
##          _id count
## 1       Lore   243
## 2 Leadership   242
## 3     Spirit   232
## 4    Tactics   232
## 5    Neutral   133
## 6 Fellowship    14
## 7    Baggins     4
```

Pipeline d'agrégation - Table de contingence

Pour des effectifs croisés, `_id` est un peu plus compliqué ...

```
m$aggregate('[
  { "$group": { "_id": { "sphere_name": "$sphere_name", "type_name": "$type_name" }
                "count": { "$sum": 1 } } }
]') %>% head(10)
```

```
##    _id.sphere_name     _id.type_name count
## 1          Tactics             Event    62
## 2             Lore Player Side Quest     2
## 3       Fellowship        Attachment     3
## 4       Leadership             Event    65
## 5       Leadership        Attachment    50
## 6          Neutral        Attachment    34
## 7          Baggins             Event     2
## 8             Lore              Hero    53
## 9           Spirit Player Side Quest     2
## 10          Spirit             Event    60
```

Pipeline d'agrégation - Opérateurs

Des opérateurs sont disponible pour calculer des résultats plus avancés.

```
m$aggregate('[
  { "$group": { "_id": "$sphere_name",
                "attack": { "$avg": "$attack" },
                "defense": { "$avg": "$defense" } } }
]')
```

```
##          _id   attack  defense
## 1     Spirit 1.105691 1.138211
## 2    Neutral 2.423077 2.192308
## 3    Baggins 1.000000 1.000000
## 4 Fellowship 1.750000 2.000000
## 5       Lore 1.300000 1.100000
## 6    Tactics 2.042373 1.338983
## 7 Leadership 1.376000 1.128000
```

Pipeline d'agrégation - Opérateurs

Des opérateurs sont disponible pour calculer des résultats plus avancés.

```
m$aggregate('[
  { "$group": { "_id": "$sphere_name",
                "attack": { "$avg": "$attack" },
                "defense": { "$avg": "$defense" } } }
]')
```

Pipeline d'agrégation - Opérateurs

Pour aller plus loin

...https://docs.mongodb.com/manual/reference/operator/aggregation/

```
m$aggregate('[
  { "$match": { "threat": { "$exists": true } } },
  { "$project": { "niveau": { "$cond": [ { "$lt": ["$threat", 10] },
                                         "Faiblard", "Balèze"] } } },
  { "$group": { "_id": "$niveau", "count": { "$sum": 1 } } }
]')
```

```
##        _id count
## 1 Faiblard   159
## 2   Balèze    60
```

Pipeline d’agrégation - Limites

Chaque document de la collection résultante est limité à 16MB : il s’agit d’une contrainte liée au format BSON utilisé dans MongoDB.

Chaque stage est limité à 100MB de RAM : cela peut poser des problèmes lorsque les jeux de données sont volumineux.

Le pipeline est limité aux opérateurs définis par MongoDB : il n’est pas possible d’utiliser des fonctions plus “souples” pour calculer le résultat d’une agrégation.

MapReduce

MongoDB propose une alternative pour réaliser des agrégations : **MapReduce**

MapReduce est un **patron d'architecture** breveté par Google en 2004. Le principe est de **paralléliser** les calculs qui peuvent alors être **distribués** et de rendre cela invisible à l'utilisateur. Pour cela, MapReduce opère en deux étapes :

- **map** : fonction appliquée à chaque document qui **émet** une sortie,
- **reduce** : les sorties des appels **map** sont regroupées pour produire le résultat.

MapReduce

Quelques remarques pour MapReduce avec MongoDB :

- la sortie d'un **map** est un document, i.e. des paires clé/valeur,
- les fonctions **map** et **reduce** sont écrites en JavaScript (pas en **R**),
- le nom de la méthode à utiliser est `mapreduce`,
- il est possible de limiter MapReduce au résultat d'une recherche avec le paramètre `query` ou de trier le résultat avec `sort`.

MapReduce offre plus de souplesse que le pipeline d'agrégation mais ce dernier doit rester le choix à privilégier car MapReduce est moins efficace et plus complexe en général.

MapReduce - Compter des effectifs

Considérons un premier exemple simple de MapReduce :

```
m$mapreduce('function() { emit(this.pack_name, 1) }',
            'function(key, values) { return Array.sum(values) }') %>% head(7)
```

```
##                          _id value
## 1      A Shadow in the East    13
## 2          The Redhorn Gate    10
## 3           The Morgul Vale    10
## 4    Escape from Mount Gram    10
## 5    The Wilds of Rhovanion    15
## 6     Encounter at Amon Dîn    10
## 7 Flight of the Stormcaller    11
```

MapReduce - Calculer des moyennes

Voici comment obtenir la longueur moyenne des noms en fonction de la Sphere :

```
m$mapreduce('function() { emit(this.sphere_name, this.name.length) }',
            'function(key, values) { return Array.sum(values) / values.length }'
)
```

```
##          _id    value
## 1       Lore 13.99177
## 2    Tactics 13.40948
## 3 Leadership 13.85537
## 4    Neutral 14.01504
## 5     Spirit 13.71121
## 6    Baggins 14.25000
## 7 Fellowship 13.57143
```