Université de Bordeaux
Master Informatique

Rapport Programmation Large échelle

Auteurs :

Sébastien MELEZAN

Mathieu Duban

Année universitaire 2020-2021

Dépôt Github :

https://github.com/mathiouu/Twitter-Ple

Table des matières

1 Questions Infrastructures

1.1 a) Combien de jours pouvons nous stocker ?

On sait qu'une ligne stockée nécessite en moyenne 5 Ko. Or nous avons en moyenne 504 millions de tweets par jour. Nous avons donc par jour en moyenne :
1 jour $\Rightarrow$ 5 * 504 000 000 = 2 520 000 000 Ko.
Nous avons une réplication de 3, donc :
2 520 000 000 * 3 = 7,56 To.

Sur notre infrastructure nous avons 20 machines contenant chacune 1 To. Donc notre infrastructure de test peut contenir 20 To. Or, pour 1 jour nous avons 7,56 To.
Nous pouvons donc par conséquent stocker 2 jours entiers de tweets sur celle-ci.

1.2 b) Combien de blocs de données en moyenne sur chaque machine ?

On sait d'après l'énonce que la taille d'un bloc vaut 128 Mo.
Or d'après la question au dessus nous avons trouvé qu'il était possible de stocker 2 jours complets. Donc :
On a 128 Mo = 128 * 10^{-6} To = 0.000128 To.
7.56 * 2 / 0.000128 = 118 125 Blocs.
Donc on aura 118 125 / 20 = 5 907 Blocs par machine.

1.3 c) Calculer le nombre de machine total pour avoir 5 ans de tweets

On sait qu'un jour de données équivaut à 7,56 To. On sait également que dans une année on a environ 365 jours. Donc :
365 * 5 * 7.56 = 13 797 To.
Au bout de 5 ans, on aura donc au total 13 797 To soit 13.797 Po. Or actuellement nous possédons 2O To sur notre infrastructure.
13 797 - 20 = 13 777 To
Il nous faudra donc par conséquent 13 777 machines pour stocker la totalité des données car nos machines font 1 To.

2 Analyse

Les différentes analyses que nous avons pu faire, ont été réalisées à l'aide de **Spark** ainsi que le webui se trouvant sur la machine **data** en salle 203.

2.1 Analyse sur les Hashtags

2.1.1 Top K Hashtags

Contexte : Pouvoir récupérer la liste des **k** hashtags les plus utilisés ainsi que leur nombre d'apparition avec $1 <= k <= 10000$.

Explication : Pour répondre à cette question nous avons utilisé le pattern *TopK*.
Dans un premier temps, nous avons un *RDD* partitionné qui stocke chaque ligne du fichier courant. Nous lui appliquons ensuite un *flatMap* afin de récupérer les hashtags et leur associer par la même occasion un compteur (initialisé à 1) pour pouvoir les compter.
Dans un second temps, nous avons un *reduceByKey* qui réduit tous les couples ayant les mêmes *key* tel que $(a, b) -> a + b$. Ce qui nous permet de compter le nombre d'occurrences d'un hashtag.
Dans un dernier temps, on récupère les K premiers hashtags.

Temps en pratique : 3min30 sur 21 fichiers.

2.1.2 Nombre d'apparition d'un hashtag

Contexte : Récupérer le nombre d'apparition d'un hashtag sur l'ensemble des données

explication : Nous partons du RDD partitionné qui contient les lignes des fichiers nljson. Nous faisons ensuite un flatMapToPair pour récupérer les hashtags comme clé et un 1 comme valeur. Tout ceci ce fait dans chaque partition donc en interne. Sur ce PairRdd nous faisons un reduceByKey qui va réunir les hashtags et ajouter 1 à chaque occurrence.

temps en pratique : 3mins, 8sec sur les 21 fichiers

2.1.3 Tous les utilisateurs qui ont utilisé un hashtag

Contexte : Récupérer tous les utilisateurs qui ont utilisés un hashtag sans doublons.

explication : Nous partons du RDD partitionné qui contient les lignes des fichiers nljson. Nous faisons ensuite un flatMapToPair pour récupérer les hashtags comme clé et un nom d'utilisateur comme valeur.
Ensuite avec un aggregateByKey nous transformons chaque paire <String,String> en paire <String,LinkedHashSet<String» pour récupérer une liste d'utilisateur par hashtag. La linkedHashSet est utilisé à la place d'une simple ArrayList pour éviter les doublons lors de l'insertion.

temps en pratique : 6mins, 35sec sur les 20 fichiers

2.2 Analyse sur les utilisateurs

2.2.1 Liste des hashtags d'un utilisateur

Contexte : Pouvoir récupérer la liste des hashtags d'un utilisateur sans doublon.

Explication : Dans un premier temps, nous avons un *RDD* qui stocke chaque ligne du fichier courant. Nous lui appliquons ensuite un *flatMap*. Le *RDD* obtenu contient un *Tuple2*. Ce tuple est composé de :

- key ⇒ Long ⇒ correspondant à l'id d'un utilisateur
- value ⇒ Tuple2<String, LinkedHashSet<String»
  - key ⇒ le *screen_ name* de l'utilisateur
  - value ⇒ la liste des hashtags utilisés par l'utilisateur sur le tweet

.
Nous stockons le nom de l'utilisateur (son @) afin de pouvoir retrouver qui il est sans devoir passer par son id. De plus, étant donné que nos données sont sur plusieurs jours il se peut que l'utilisateur ait changé de nom entre temps, donc il nous faut tous les stocker.
Dans un second temps, afin de transformer le *RDD* en *PairRDD* nous utilisons la fonction *mapToPair*.
Dans un troisième temps, nous appelons la fonction *getUsersHashtags* (implémentée par nous) qui permet de récupérer un *PairRDD*<String, String> avec la *key* étant le dernier *screen_ name* utilisé par la personne et la *value* étant la liste des hashtags sans doublon utilisé par l'utilisateur.

Dans cette fonction, nous utilisons un *reduceByKey*(a,b) qui nous permet pour chaque id d'utilisateur, faire un *String* tel que "screenNameA, screenNameB" et une *LinkedHashSet* qui permet de grouper les 2 listes d'hashtags sans doublon. Nous retournons ensuite un tuple composé de ces deux éléments.

Suite à cela, nous utilisons un *mapToPair* qui nous permet de transformer notre *JavaPairRDD<Long, Tuple2<String, LinkedHashSet<String»>* obtenu en *JavaPairRDD<String, String>*. Nous récupérons en key le dernier nom de l'utilisateur (comme expliqué précédemment) et nous transformons le *LinkedHashSet* en un *String* tel que "hashtags0, hashtags1, ..., hashtagsX".

Ensuite nous enregistrons notre *PairRDD* dans **HBase** en passant par un *SequenceFile* en utilisant un map-reducer d'hadoop. (fonction *fillHbaseWithSequenceFile*).

Temps en pratique : 4min09s pour les 21 fichiers.

2.2.2 Permettre d'avoir le nombre de tweet d'un utilisateur

Contexte : Pouvoir récupérer le nombre de tweet fait par un utilisateur.

Explication : Dans un premier temps, nous avons un *RDD* qui stocke chaque ligne du fichier courant. Nous lui appliquons ensuite un *flatMap*. Le *RDD* obtenu contient un *Tuple2*. Ce tuple est composé de :

- key ⇒ Long ⇒ correspondant à l'id d'un utilisateur
- value ⇒ Tuple2<String, Integer>
  - key ⇒ le *screen_ name* de l'utilisateur
  - value ⇒ valeur qui vaut 1 permettant de compter les tweets

.
Nous stockons le nom de l'utilisateur (son @) afin de pouvoir retrouver qui il est sans devoir passer par son id. De plus, étant donné que nos données sont sur plusieurs jours il se peut que l'utilisateur ait changé de nom entre temps, donc il nous faut tous les stocker.
Dans un second temps, afin de transformer le *RDD* en *PairRDD* nous utilisons la fonction *mapToPair*.

Dans un troisième temps, nous appelons la fonction *getUsersNbTweet* (implémentée par nous) qui permet de récupérer un *PairRDD*<String, Integer> avec la *key* étant le dernier *screen_ name* utilisé par la personne et la *value* étant le nombre de tweet d'un utilisateur.

Dans cette fonction, nous utilisons un *reduceByKey*(a,b) qui nous permet pour chaque id d'utilisateur, faire un *String* tel que "screenNameA, screenNameB" et un *Integer* qui permet d'additionner les valeurs comptant le nombre de tweet.

Suite à cela, nous utilisons un *mapToPair* qui nous permet de transformer notre *JavaPairRDD<Long, Tuple2<String, Integer»* obtenu en *JavaPairRDD<String, Integer>*. Nous récupérons en key le dernier nom de l'utilisateur (comme expliqué précédemment) et nous retournons la valeur qui contient le nombre de tweet de la personne.

Ensuite nous enregistrons notre *PairRDD* dans **HBase** en passant par un *SequenceFile* en utilisant un map-reducer d'hadoop. (fonction *fillHbaseWithSequenceFile*).

Temps en pratique : 5min sur tous les fichiers.

2.2.3 Remarques

Permettre de récupérer pour chaque utilisateur la liste de ses hashtags sans doublon et permettre d'avoir le nombre de tweet qu'il a fait correspond à peu près la même implémentation.

2.2.4 Nombre de tweet par pays

Contexte : Pouvoir récupérer le nombre de tweet par pays

Explication : Dans un premier temps, nous avons un *RDD* qui stocke chaque ligne du fichier courant. Nous lui appliquons ensuite un *flatMap*. Le *RDD* obtenu contient un *Tuple2*. Ce tuple est composé de :

— key ⇒ String ⇒ correspondant au nom du pays

— value ⇒ Integer ⇒ valeur qui vaut 1 permettant de compter les tweets

.
Dans un second temps, afin de transformer le *RDD* en *PairRDD* nous utilisons la fonction *mapToPair*.

Dans un troisième temps, nous utilisons un *reduceByKey*(a,b) permettant d'additionner les tweets entre eux par pays.

Ensuite nous enregistrons notre *PairRDD* dans **HBase** en passant par un *SequenceFile* en utilisant un map-reducer d'hadoop. (fonction *fillHbaseWithSequenceFile*).

Temps en pratique : à déterminer

2.2.5 Nombre de tweet par langue

Contexte : Pouvoir récupérer le nombre de tweet par langue

Explication : Dans un premier temps, nous avons un *RDD* qui stocke chaque ligne du fichier courant. Nous lui appliquons ensuite un *flatMap*. Le *RDD* obtenu contient un *Tuple2*. Ce tuple est composé de :

— key ⇒ String ⇒ correspondant à la langue (nous utilisons notre classe *Lang* qui permet de parser la langue car nous récupérons une abréviation de la langue. Nous la transformons avec la langue correspondante.

— value ⇒ Integer ⇒ valeur qui vaut 1 permettant de compter les tweets

.
Dans un second temps, afin de transformer le *RDD* en *PairRDD* nous utilisons la fonction *mapToPair*.

Dans un troisième temps, nous utilisons un *reduceByKey*(a,b) permettant d'additionner les tweets entre eux par langue.

Ensuite nous enregistrons notre *PairRDD* dans **HBase** en passant par un *SequenceFile* en utilisant un map-reducer d'hadoop. (fonction *fillHbaseWithSequenceFile*).

Temps en pratique : à déterminer

2.2.6 Remarques

Permettre de récupérer le nombre de tweet par pays ou par langue correspond à peu près la même implémentation.

2.3 Influenceurs

2.3.1 Triplets de hashtags ainsi que leurs utilisateurs

Contexte : Récupérer les triplets de hashtags, leur nombre d'apparition et les utilisateurs qui les ont utilisés sans doublons. Récupérer aussi le nombre de fois qu'un utilisateur à tweeter avec ce triplet de hashtag.

Explication : C'est celui qui nous a posé le plus problème, actuellement nous le faisons tourner avec 8 fichiers uniquement.
Grace au premier flatMapToPair nous avons récupéré comme clé, le triplet de hashtag, et comme valeur, un tuple2 contenant la liste des utilisateurs et un entier qui va nous servir de compteur.
Dans un second temps nous réduisons pour faire réunir tous les utilisateurs et augmenter le compteur. Pour l'instant nous gardons délibérément les doublons d'utilisateurs. Ensuite le MapToPair va compter les doublons et les insérer dans un HashMap afin d'avoir un username en tant que clé et le nombre de fois qu'il est apparu en tant que valeur.
Nous effectuons un tri et nous utilisons un cache afin de ne pas tout recalculer. D'un côté nous insérons dans la première table pour récupérer tous les triplets, de l'autre côté nous récupérons uniquement les utilisateurs et le triplet en tant que clé et le nombre de fois qu'ils l'ont utilisés en tant que valeur, avant de l'insérer dans la deuxième table.

temps en pratique : 3mins, 28sec

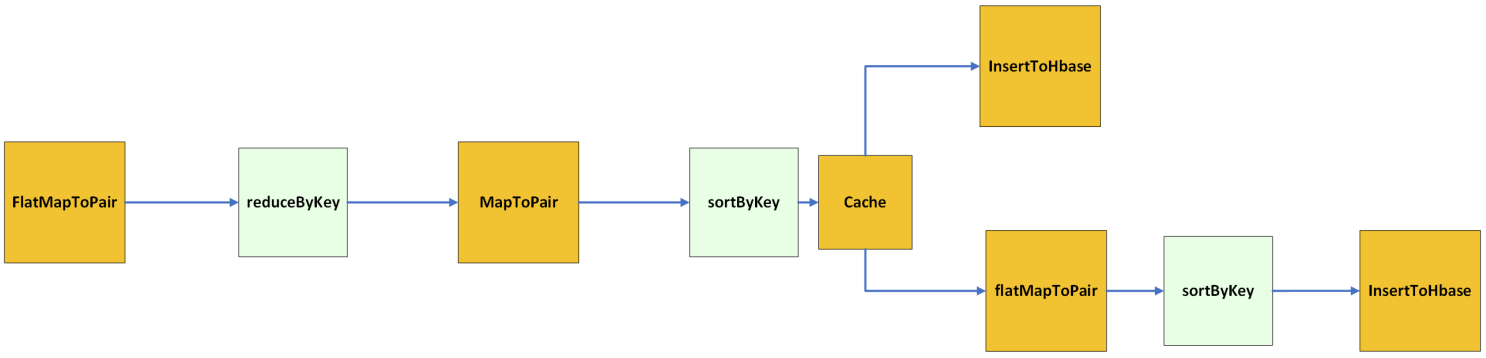

FIGURE 1 – Schéma

3 Documentation

Ne pouvant pas utiliser **Docker** car le CREMI ne le veut pas, il faut ouvrir plusieurs terminaux afin de lancer divers commandes.

3.1 Lancement de l'application

3.1.1 Back

Pour lancer la partie **Back**, il faut dans un premier temps faire *npm install* dans */app/server*.

Dans un second temps, il faut lancer *hbase rest start* afin d'accéder aux différentes bases de données, dans un terminal et le laisser tourner.

Dans un troisième temps, il faut lancer *npm run start* afin de lancer la partie **Back**.

3.1.2 Front

Pour lancer la partie **Front**, il faut dans un premier temps faire *npm install* dans *app/client/front*.

Dans un second temps, il faut lancer *npm run serve* afin de lancer la partie **Front**.

Une fois cela fait, il suffit de cliquer ici afin d'avoir notre application web (partie client). Vous pouvez retrouver la partie **Back** ici (partie serveur).