TP Hadoop : Introduction au Map Reduce

Mastère Big Data 2019/2020

Benjamin Thery - benjamin.thery@grenoble-inp.org

1. Prise en main

1.1 Execution locale

Question 1. Que signifie *Map input records* ? Et *Map output records* ?

Map input record : nombre de lignes du fichier d'entrée en entrée du Mapper.
Map output record : nombre d'enregistrements produits en sortie du Mapper.

Question 2. Quel est le lien entre *Map output records* et *Reduce input records* ?

Leur valeur est la même.
Les *Map output records* sont les enregistrements qui vont servir d'entrée aux *Reducers*.

Question 3. Que signifie *Reduce input groups* ?

C'est le nombre de clés uniques (mots uniques) passées en entrée des *Reducers*.

1.2 Premier contact avec HDFS

Connection au serveur Hadoop : ssh theryb@152.77.81.30

Commandes HDFS :

```
$ hdfs dfs
```

```
$ hdfs dfs -ls /
Found 3 items
drwxr-xr-x   - hduser   hdgroup          0 2019-11-26 11:46 /data
drwxrwxrwt   - hduser   hdgroup          0 2019-11-18 11:13 /tmp
drwxr-xr-x   - hduser   hdgroup          0 2019-11-14 12:14 /user
```

```
$ hdfs dfs -ls /user
...
drwxr-xr-x   - theryb   theryb              0 2019-10-21 14:57 /user/theryb
...
```

```
$ hdfs dfs -ls /user/theryb
```

Question 1. Quel est le chemin, dans HDFS, vers votre répertoire personnel ?

Le chemin d'accès vers le répertoire personnel est **/user/theryb**

1.3 Exécution sur le cluster

Compter le nombre d'occurrences de chaque mot dans les 5 tomes des Misérables de Victor Hugo.

Question 1. A quel compteur le nombre de *splits* correspond-il ?

Le nombre de splits correspond au compteur ***Launched map tasks*** quand aucune tache n'a échoué,
ou à *Launched map tasks - Failed map tasks* quand certaine taches ont échoué et ont du être relancées.

1.4 Combiner

Ajouter un combiner :

```
job.setCombinerClass(WordCountReducer.class);
```

Dans le programme proposé, on ne peut réutiliser directement la classe WordCountReducer telle quelle car ces types d'entrée ne correspondent pas aux types de sortie (int vs long). Une nouvelle classe **WordCountCombiner** similaire WordCountReducer a été ajoutée mais avec une signature différente :

```
public static class WordCountCombiner extends Reducer<Text, IntWritable, Text,
IntWritable> { … }
```

Fichier source : WordCount/WordCountWithCombiner.java

Question 1. Quels compteurs permettent de vérifier que le combiner a fonctionné ?

Les compteurs ***Combine input records*** et ***Combine output records*** permettent de vérifier que le combiner à fonctionné. Leur valeur est non nulle lors de l'exécution avec le combiner.

Question 2. Quels compteurs permettent d'estimer le gain effectivement apporté par le combiner ? Comparez aux valeurs obtenues sans combiner pour justifier votre réponse.

Diff des compteurs de sortie entre l'exécution sans le combiner (-) et avec le combiner (+) :

```
-        Combine input records=0
-        Combine output records=0
+        Combine input records=401392
+        Combine output records=55487

         Reduce input groups=29373
-        Reduce shuffle bytes=4677667
-        Reduce input records=401392
+        Reduce shuffle bytes=800835
+        Reduce input records=55487
```

Les compteurs ***Reduce input records*** et ***Reduce shuffle bytes*** permettent d'estimer le gain apporté par le combiner.
Reduce input records est beaucoup plus petit dans le cas où un combiner a été utilisé : ~55'000 < ~400'000. Beaucoup moins d'enregistrements ont été poussés en entrée du reducer (seulement 14%).

Question 3. Quel est le mot le plus utilisé dans Les Misérables ?

Le mot le plus utilisé dans les Misérables est **de** avec 16955 occurrences**.**

```
$ sort -r -n -k 2 part-r-00000  | head -10
de    16955
la    12253
et    10979
le    9810
il    7660
les   6059
un    5112
que   4430
dans  4235
qui   4225
```

1.5 Nombres de reducers

Utiliser 3 reducers :

```
job.setNumReduceTasks(3)
```

Source file : WordCount/WordCountWithCombiner.java

Question 1. Quelle est la différence entre le répertoire de résultats obtenu ici, et celui de la partie 1.3 ? Pourquoi ?

Le répertoire de résultats contient maintenant 3 fichiers de sortie, un par reducer.
Chaque reducer a généré ses résultats sur le sous ensemble de clés qui lui a été assigné.

```
$ ls
part-r-00000  part-r-00001  part-r-00002  _SUCCESS
```

Le résultat global peut-être retrouvé en concaténant les fichiers résultat de tous les reducers :

```
$ cat part-r-00000 part-r-00001 part-r-00002 | sort -r -n -k 2 | head -5
de    16955
la    12253
et    10979
le    9810
il    7660
```

1.6 In-Mapper Combiner

Ajout d'un *in-mapper combiner* à l'aide d'une table de hashage.

Fichier source : WordCount/WordCountWithInMapperCombiner.java

1.7 Compteur

Ajout d'un compteur pour enregistrer le nombre de lignes vides dans les données.

Fichier source : WordCount/WordCountWithInMapperCombinerWithCounter.java

```
INFO  LocalJobRunner - reduce > reduce
INFO  Task - Task 'attempt_local1212808654_0001_r_000000_0' done.
```

```
INFO  Job - Job job_local1212808654_0001 running in uber mode : false
INFO  Job -  map 100% reduce 100%
INFO  Job - Job job_local1212808654_0001 completed successfully
INFO  Job - Counters: 28
      File System Counters
      ...
      Map-Reduce Framework
      ...
      WordCountWithInMapperCombinerWithCounter$WordCountMapper$CounterType
            EmptyLine=3
      File Input Format Counters
            Bytes Read=111
      File Output Format Counters
            Bytes Written=47
```

2. Top-tags Flickr par pays, avec tri en mémoire

2.1 Map et Reduce

Fichier source : MostUsedTags/MostUsedTagsPerCountry.java

Résultats pour les 5 tags les plus utilisés par pays du fichier flickrSample.txt:

Sortie avec tags aggrégés par pays par le reducer:

```
$ cat out/part-r-00000
AG          (3)الطوارق, algeria(3), amazigh culture(3), (3)تمنراست, hoggar(3)
BN          ghana(7), lab(5), africa(2), idds(2), single mothers(1)
ML          mali(15), niger(11), islam(10), rio niger(10), desierto(10)
UV          africa(10), burkina-faso(9), afrique(9), burkina faso(9), ghana(8)
Unknown     rojer(6), orangevale(6), rojer wisner(6), rww(6), roger beach(6)
```

Sortie brute :

```
$ cat out/part-r-00000
AG    الطوارق
AG    algeria
AG    amazigh culture
AG    تمنراست
AG    الهقار
BN    ghana
BN    lab
BN    africa
BN    idds
BN    single mothers
ML    mali
ML    niger
ML    islam
ML    viajes
ML    rio niger
UV    africa
UV    burkina-faso
UV    afrique
UV    burkina faso
UV    ghana
Unknown     rojer
Unknown     orangevale
Unknown     rojer wisner
Unknown     rww
Unknown     roger beach
```

Notes:

- *Dans le jeu de données, certaines entrées n'ont pas de tags, elles sont exclues par le mapper. J'ai ajouté un compteur Hadoop pour les compter.*
- *La classe Country ne trouve pas de pays pour certaines coordonnées présentent dans le fichier flickrSample.txt (par exemple (-1.0,-1.0)). Je les ai regroupées dans un pays « Unknown ». Il y aussi un compteur Hadoop pour les compter.*
- *Dans la sortie avec tags aggrégés, les chaines contenant des tags en langue arabe sont écrits en RTL par Java (right-to-left), le nombre d'occurrences du tag se retrouve à gauche.*

2.2 Combiner

Fichier source : MostUsedTags/MostUsedTagsPerCountryWithCombiner.java

Question 1: Pour pouvoir utiliser un combiner, quel devrait être le type des données intermédiaires ? Donnez le type sémantique (que représentent ces clés-valeurs ?) et le type Java.

Pour pouvoir utiliser un combiner le type de données intermédiaires devrait être de type:

- Clé : **Text** (code pays)
- Valeur : **(Text, Int)** (tag, nombre d'occurrence du tag)
  - Cela permet de commencer à compter le nombre d'occurrence de chaque tag d'un pays dans le combiner et de passer cette première agrégation au reducer.
  - En Java, on peut réutiliser la classe *StringAndInt* pour représenter le tuple (Text, Int) qui représente un tag et son nombre d'occurrences.

Mapper :

- Input : (Long, Text)
- Ouput : (Text, (Text, Int))
- Signature Java :

```
public static class MostUsedTagsMapper
      extends Mapper<LongWritable, Text, Text, StringAndIntWritable> {}
```

Combiner :

- Input : (Text, (Text, Int))
- Ouput : (Text, (Text, Int))
- Signature Java :

```
public static class MostUsedTagsCombiner
      extends Reducer<Text, StringAndIntWritable, Text, StringAndIntWritable> {}
```

Reducer :

- Input : (Text, (Text, Int))
- Ouput : (Text, Text)
- Signature Java :

```
public static class MostUsedTagsReducer
      extends Reducer<Text, StringAndIntWritable, Text, Text> {}
```

Question 2 : Quels sont les tags les plus utilisés en France ?

Les 5 tags les plus utilisés pour la France :

```
FR    france(35392), paris(24254), barcelona(12468), spain(9779), europe(6170)
```

Les tags 'barcelona' et 'spain' apparaissent probablement car l'algorithme de la classe Country doit considèrer que les coordonnées de Barcelone sont plus proches du centre de la France que de celui de l'Espagne.

Note : Sur le jeu de donnée /data/flickr.txt, contenant 20 millions d'enregistrements, mon programme en a trouvé 8'033'752 sans tags et 9'866'381 sans coordonnées (ou pour lesquelles la classe Country ne retourne pas de pays).

Question 3 : Dans le reducer, nous avons une structure en mémoire dont la taille dépend du nombre de tags distincts : on ne le connaît pas a priori, et il y en a potentiellement beaucoup. Est-ce un problème ?

Oui, c'est un problème. Dans un cas extrême avec un très grand nombre de tags, la hashmap utilisée pour compter les occurrences de chaque tags associés à un pays pourrait ne pas tenir dans la mémoire allouée au reducer. La réduction pourrait ne pas être exécutée.

3. Top-tags Flickr par pays, avec mémoire limitée