Introduction à Map-Reduce - TP

Martin Kirchgessner, Vincent Leroy

2017

Votre compte-rendu sera une archive `Votre_Nom.zip` (ou `tar.gz`) contenant :
— Un fichier avec les réponses aux questions qui suivent, et
— Vos sources. Ecrivez un fichier par problème qui vous est soumis. Exemple : `Question3_1.java` pour la question 3.1.

Envoyez l'archive à Vincent.Leroy@univ-grenoble-alpes.fr, avec [TPHadoopISI] au début du sujet. Mettez votre binôme en copie de cet e-mail.

0 Kit de survie sous Eclipse

Usez et abusez des raccourcis clavier suivants :
— *Ctrl + Espace* pour l'autocomplétion, qui permet aussi de générer des méthodes (essayez dans le corps d'une classe)
— *Ctrl + Shift + 1* "Quick fix", propose des actions pertinentes selon ce qu'il y a sous le curseur.
— *Ctrl + Shift + O* pour ajouter les imports manquants et retirer les inutiles.
— *Ctrl + clic* pour atteindre la définition du nom sous le curseur de la souris (utilisez *Alt* + ← pour revenir là où vous étiez).
— *Alt + Shift + R* pour renommer ce qui est sous le curseur (partout où c'est pertinent).

Voir aussi `http://eclipse-tools.sourceforge.net/Keyboard_shortcuts_(3.0).pdf`

1 Prise en main

Nous avons préparé un projet Eclipse qui contient les librairies nécessaires. Vous pourrez ainsi tester vos programmes localement, sur de petits fichiers, avant de les lancer sur le cluster. Ce qui est très pratique si vous avez besoin de debugger avec des points d'arrêt.

1. Téléchargez sur votre poste de travail `TPIntroHadoop.zip`
2. Dans les menus d'Eclipse, cliquez sur `File > Import ...`
3. Dans la catégorie `General`, sélectionnez `Existing Projects into Workspace` (et non `Archive`)
4. Choisissez `select archive file` et retrouvez l'archive téléchargée
5. Cliquez sur `Finish`

Le répertoire des sources contient `Question0_0.java`, qui peut vous servir d'exemple pour écrire chaque programme (mais pour certaines questions, attention à modifier les types de clés/valeurs qu'il déclare). Vous remarquerez que le `Mapper` et `Reducer` sont déclarés comme des classes internes : ce n'est pas imposé par le framework, c'est même déconseillé. On se le permet dans le cadre du TP pour faciliter les corrections.

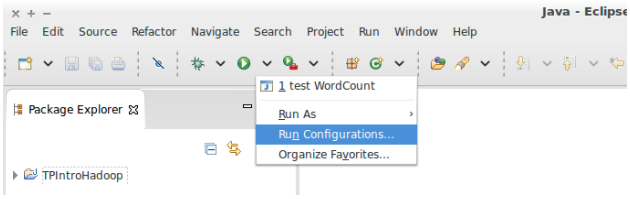

FIGURE 1 – Pour indiquer quels paramètres utiliser en test local, il faut créer une configuration d'éxecution via le menu du bouton "Run"

1.1 Exécution locale

Avant de lancer un programme MapReduce sur un cluster, on le teste *toujours* localement, sur un échantillon des données à traiter. Dans ce cas, le framework ne lit/écrit pas sur HDFS, mais sur votre système de fichiers local.

1. Créez un programme qui compte le nombre d'occurrences de chaque mot dans un fichier texte (cf. "WordCount" vu en cours).
2. Créez à la racine du projet un fichier texte, avec le (petit) contenu de votre choix.
3. La fonction `main` prend en argument le chemin du fichier d'entrée et de sortie. Avant de lancer le programme avec Eclipse, il faut donc lui indiquer ces chemins (vous pouvez utiliser des chemins relatifs à la racine du projet) - cf. Figure 1.

Lancez le programme. Les traces d'éxécution du framework apparaîssent dans l'onglet "Console". En cas de bug, vous y verrez l'exception à l'origine du plantage. Ouvrez le fichier généré et vérifiez vos résultats.

Quand le programme marche, à la fin des traces d'éxécution se trouvent les compteurs tenus à jour par Hadoop. Ajoutez/retirez du contenu à votre fichier de texte, relancez votre programme et observez l'évolution des compteurs pour répondre aux questions suivantes.

1. A quoi correspondent les compteurs `Map input records` ? Et `Map output records` ?
2. Quel est le lien entre `Map output records` et `Reduce input records` ?
3. A quoi correspond `Reduce input groups` ?

1.2 Combiner

Nous allons modifier légèrement le programme, et comparer les compteurs et résultats.

— Ajoutez `job.setCombinerClass(???)` dans le programme principal (à vous de compléter avec la bonne valeur en argument).

— Vérifiez que le programme fonctionne localement, puis éxécutez-le sur les 5 tomes des *Misérables*

Questions :

1. Quels compteurs permettent de vérifier que le combiner a fonctionné ?
2. Quels compteurs permettent d'estimer le gain effectivement apporté par le combiner ? Comparez aux valeurs obtenues sans combiner pour justifier votre réponse.

2 Top-tags Flickr par pays, en un job

Nous allons maintenant analyser des meta-données de photos, en utilisant un extrait du jeu de données "Yahoo ! Flickr Creative Commons 100M". Le but est de trouver les tags les plus utilisés par pays. Pour identifier un pays à partir des coordonnées d'une photo, vous utiliserez la classe fournie `Country`.

L'archive que vous avez téléchargée au début du TP contient deux fichiers qui vont vous aider à vous familiariser avec ces donneés :

- `flickrSpecs.txt` décrit le format du fichier. Pour décoder les textes, par exemple les tags, utilisez `java.net.URLDecoder.decode(String s)`,
- `flickrSample.txt` est un extrait à utiliser pour les tests locaux.

A partir d'ici, la référence de l'API Hadoop pourra vous être utile :
`https://hadoop.apache.org/docs/r2.2.0/api/index.html`

Attention : dans cette référence, de nombreuses classes existent en double. Quand c'est le cas, choisissez toujours celle qui appartient au package `org.apache.hadoop.mapreduce`, qui correpond à l'API moderne, dite "YARN". Les doublons appartiennent au package `org.apache.hadoop.mapred`, qui est celui de l'ancienne API, dite "MR1".

2.1 Map et Reduce

Puisque l'on veut trouver les K tags les plus utilisés *par pays*, a priori l'identifiant d'un pays (les deux lettres retournées par `country.toString()`) est une bonne clé intermédiaire. Les valeurs associées seront les tags (les chaines de caractères) qui ont été associés à une photo prise dans ce pays. En sortie, il ne restera qu'au plus K couples $(pays, tag)$ par pays.

A vous d'implémenter le programme complet. Pour la fonction `reduce` :

1. Comptez le nombre d'occurrences de chaque tag avec `java.util.HashMap<String,Integer>`.
2. Puis triez les tags par nombre d'occurrences décroissants :
   - Créez (dans un fichier séparé) une classe `StringAndInt` qui implémente l'interface `Comparable<StringAndInt>` et contient deux champs, qui dans ce cas représenteront le tag et son nombre d'occurrences. La méthode `compareTo` ne prendra en compte que le nombre d'occurrences.
   - Utilisez la classe `java.util.PriorityQueue<StringAndInt>` afin de ne conserver que les K tags les plus populaires.

K sera un nouveau paramètre à passer au programme principal via la ligne de commande. Utilisez l'objet `Configuration` pour transmettre sa valeur du programme principal aux *reducers* (via l'objet `Context`).

Implémentez puis testez *localement* ce programme.

2.2 Combiner

Question : 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.

Avant de tester ce programme sur le cluster :

1. Modifiez la classe que l'on a ajouté en 2.1, de sorte qu'elle implémente aussi l'interface `Writable`. La référence de cette interface vous fournira quelques indices, toutefois :
   — encapsulez un `Text` pour la chaine de caractères
   — dans leur exemple, la méthode statique `read` est inutile
   — dans leur exemple, il manque un constructeur sans arguments
2. Modifiez une copie du programme écrit dans la partie 2.1, en y ajoutant un *combiner*. Cette fois, on ne peut pas réutiliser le *reducer*, il faut implémenter une troisième classe (qui doit étendre `Reducer<K,V,K,V>`).

Executez cette variante (avec $K = 5$) sur `/data/flickr.txt`. Quels sont les tags les plus utilisés en France ?

Question : 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 ?