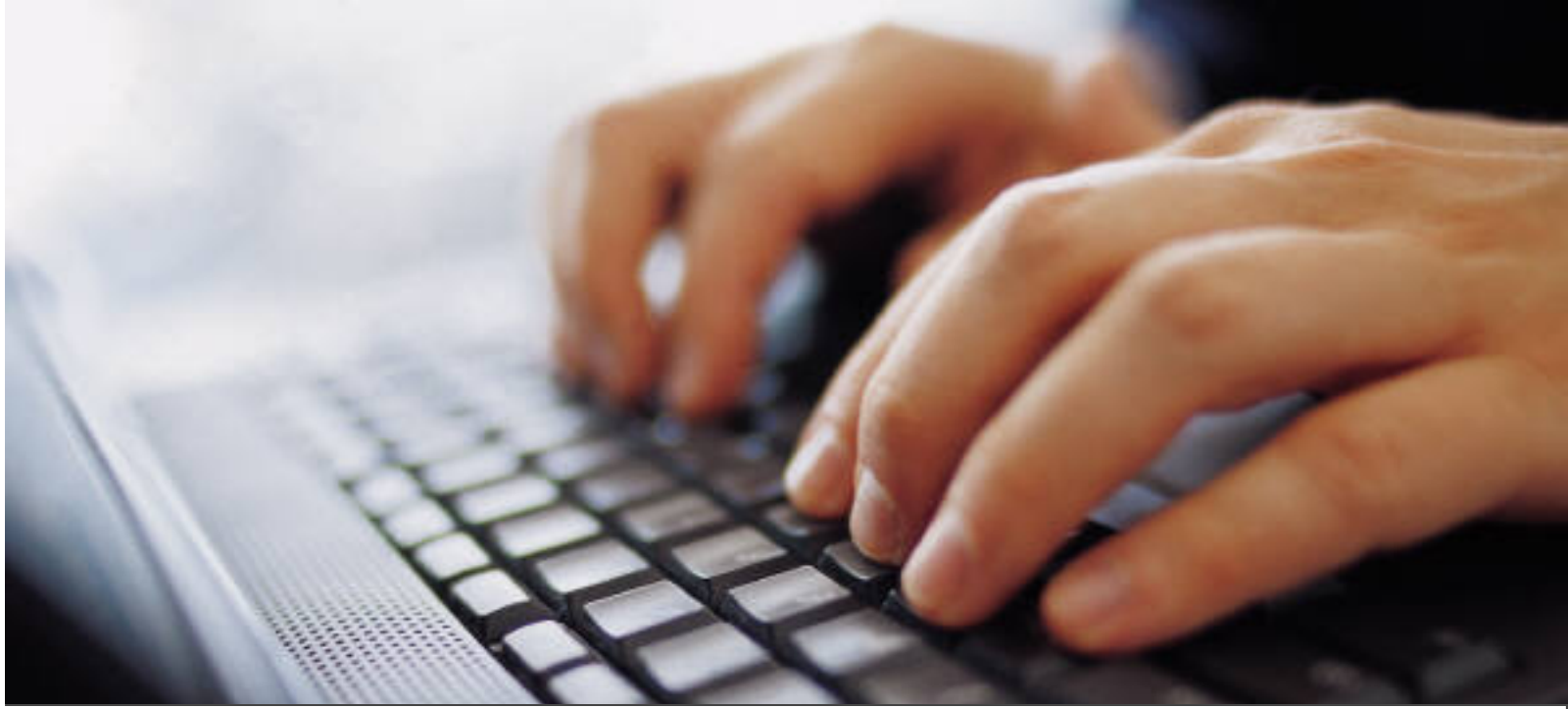
Hadoop / Big Data

Benjamin Renaut <renaut.benjamin@tokidev.fr>

MBDS

2019 – 2020



TP 1

Méthodologie Map/Reduce - programmation Hadoop.

Préparation du TP

- **Installer VirtualBox (https://www.virtualbox.org/).**
- **Importer la machine virtuelle .ova du TP.**
  Il s'agit d'un système GNU/Linux Debian, amd64.
- **Démarrer la machine virtuelle.**
- **S'identifier avec le login: `mbds`, mot de passe: `password`. Lisez le document explicatif de l'environnement pour vous connecter avec PuTTY / SSH.**
- **Démarrer Hadoop avec la commande:**

  ```
  start-hadoop
  ```

  (ignorer les éventuels *warnings*)

Préparation du TP

- **Vérifier le bon fonctionnement de Hadoop en exécutant la commande:**

  ```
  hdfs dfsadmin -report
  ```

  La commande vérifie le statut de HDFS. Elle devrait afficher, entre autres:

  ```
  Live datanodes (1)
  ```

  …suivi d’une série d’informations sur le datanode en question.

- **Nous allons maintenant compiler le code d'exemple Java du cours (compteur d’occurrence de mots).**
  Objectif: vérifier que l'environnement de développement/compilation soit fonctionnel et que Hadoop soit correctement à même d'exécuter des tâches, et se familiariser avec le processus de compilation.

Préparation du TP

- **Pour éxecuter vos programmes Hadoop Java, il faudra générer un .jar, puis le copier sur la machine virtuelle pour l'exécuter sur Hadoop.**

- **Pour le développement lui-même et la génération du .jar, Java 1.8 doit être utilisé. Vous avez par ailleurs plusieurs possibilités:**

  - **Utiliser IntelliJ, et charger les dépendances Hadoop avec Gradle. Un projet type vous est fourni pour le charger.**
  - **Utiliser IntelliJ, et charger les dépendances Hadoop avec Maven. Un projet type à importer vous est également fourni.**
  - **Utiliser Eclipse, et charger les dépendances avec Maven. Là aussi, un projet est fourni.**
  - **Utiliser un autre IDE, et charger les dépendances manuellement ou avec Maven/Gradle.**
  - **Compiler manuellement les jars sur la machine virtuelle.**

Préparation du TP

- **Si vous souhaitez utiliser IntelliJ avec Gradle, téléchargez et importez le projet suivant sur votre machine: http://cours.tokidev.fr/bigdata/tps/hadoop_intellij_project_gradle.zip**

- **Si vous souhaitez utiliser IntelliJ avec Maven, téléchargez et importez le projet suivant sur votre machine: http://cours.tokidev.fr/bigdata/tps/hadoop_intellij_project_maven.zip**

- **Si vous souhaitez utiliser Eclipse avec Maven, téléchargez et importez le projet suivant sur votre machine: http://cours.tokidev.fr/bigdata/tps/hadoop_eclipse_project_maven.zip**

- **Si vous choisissez une de ces trois options, vous pouvez sauter les slides suivantes.**

Préparation du TP

- **Si vous souhaitez utiliser un autre IDE, vous pouvez télécharger le code source Java à compiler ici: http://cours.tokidev.fr/bigdata/tps/wordcount.zip**

- **Et pour charger les dépendances Hadoop:**

  - **Utiliser la configuration Gradle suivante: http://cours.tokidev.fr/bigdata/tps/build.gradle.txt**

  - **Ou bien utiliser la configuration Maven suivante: http://cours.tokidev.fr/bigdata/tps/pom.xml.txt**

  - **Ou enfin, charger manuellement tous les .jar de cette archive comme dépendances de votre projet: http://cours.tokidev.fr/bigdata/tps/hadoop_3.1.3_deps.zip**

Préparation du TP

- **Enfin, si vous souhaitez compiler manuellement le code sur la machine virtuelle Hadoop, vous devrez y copier votre code aprés édition ou éditer le code directement dans la VM.**

- **Une fois le code présent sur la machine virtuelle, vous pouvez le compiler et construire le .jar manuellement avant de l'exécuter; en tenant compte du classpath de votre code source.**

- **A titre d'exemple, pour compiler et construire le .jar de ce premier exemple (le compteur d'occurence de mots) dans la machine virtuelle:**

  - **Télécharger le code**
    ```
    wget http://cours.tokidev.fr/bigdata/tps/wordcount.zip
    ```
  - **Le décompresser**
    ```
    unzip wordcount.zip && rm -rf wordcount.zip
    ```

Préparation du TP

- Compiler le code

```
javac WCount.java WCountMap.java WcountReduce.java
```

- Construire la hierarchie du .jar et y déplacer le code compilé

```
mkdir -p org/mbds/hadoop/wordcount
mv WCount*.class org/mbds/hadoop/wordcount/
```

- Et enfin générer le .jar:

```
jar -cvf mbds_wcount.jar -C . org
rm -rf org
```

Préparation du TP

- **Quelquesoit l'approche choisie, vous devriez désormais être à même de compiler le projet d'exemple; ce projet servira également de base pour de futurs développements.**

- **Le projet est celui vu en cours: le compteur d'occurences de mots.**

- **La première étape de ce TP consiste à compiler ce code en .jar, puis à copier ce .jar sur la machine virtuelle et à l'exécuter sur le poeme d'exemple vu en cours. L'idée ici est d'introduire le mode de travail qui sera utilisé dans un second temps pour un développement à effectuer vous-même.**

- **Commencez par compiler le projet en .jar et copier ce .jar sur la machine virtuelle. Référez-vous au document explicatif de l'environnement des TPs pour la copie.**

Préparation du TP

- **Une fois votre code compilé et le jar copié, vous allez devoir télécharger le poeme sur la machine virtuelle, le déposer sur HDFS, puis exécuter le programme en lui indiquant la position du poeme sur HDFS.**

- **On suppose que votre jar a pour nom de fichier « wordcount.jar » ; adaptez les commandes qui suivent au besoin.**

- **Commencez (dans la machine virtuelle) par télécharger le poeme à l'aide de la commande:**
  `wget http://cours.tokidev.fr/bigdata/tps/poeme.txt`

- **Puis déplacez ce fichier sur HDFS:**
  `hadoop fs -put poeme.txt /`

Préparation du TP

- **Vérifiez sa présence sur HDFS:**
  `hadoop fs -ls /`
  (le fichier « poeme.txt » doit apparaître)

- **Enfin, exécutez votre programme:**

  `hadoop jar wordcount.jar org.mbds.hadoop.tp.WCount /poeme.txt /results`

  (ajustez le nom de fichier et la classe si nécessaire; notez qu'on stocke ici les résultats dans /results sur HDFS, et gardez en tête que si ce répertoire existe déjà, le programme échouera)

- **On peut alors consulter les résultats aprés l'exécution, en faisant par exemple:**
  `hadoop fs -ls /results`
  `hadoop fs -cat /results/*`

TP - Anagrammes

- **Vous devez maintenant réaliser votre propre programme Hadoop.**
- **<u>OBJECTIF:</u> on dispose d'une liste de mots courants de la langue anglaise (plus de 1000). On souhaite déterminer quels mots sont des anagrammes.**

  On rappelle qu'un mot est un anagramme d'un autre si leurs lettres sont identiques (par exemple, « *lemon* » et « *melon* »).
- **Télécharger le fichier des mots courants avec la commande:**

```
wget http://cours.tokidev.fr/bigdata/tps/common_words_en_subset.txt
```

- **<u>Remarque:</u> vous pouvez utiliser Streaming (tel que vu en cours) si vous préférez développer dans un autre langage que Java. L'essentiel est de réaliser le programme Hadoop :-)**

Conseils

- **Pensez bien à appliquer la méthodologie vue en cours.**

  Les questions à se poser: quelle est la clef à utiliser ?
  Quel est le facteur commun entre les anagrammes, c'est à dire la clef qui permettra à Hadoop de les grouper ?

- **Ne pas oublier de déplacer les données d'entrée (le fichier des mots courants) sur HDFS.**

TP – Analyse de ventes

- **Vous devez maintenant réaliser un second programme Map/Reduce Hadoop.**

- **Il s'agit ici de travailler sur une analyse d'un receuil de ventes; on vous fourni le fichier suivant:**

  http://cours.tokidev.fr/bigdata/tps/sales_world_10k.csv

- **Il s'agit d'un fichier CSV. Notez bien son format; notez également qu'il comporte une ligne de titre (ce qui pourrait s'avérer problématique si vous le chargez dans votre programme Hadoop).**

- **On s'intéresse à cinq colonnes particulièrement: la région de vente (« Region »), le pays de vente (« Country »), le type de produit acheté (« Item Type »), le canal de vente (« Sales channel », online ou offline) et enfin le profit de vente (« Total profit »).**

TP – Analyse de ventes

- **Votre programme doit être capable de réaliser diverses tâches d'analyse.**

- **Il est fortement préférable de faire un seul programme unique d'analyse de ventes, qui soit capable de prendre en paramètre le type d'analyse à réaliser; vous pouvez ainsi faire varier le comportement du programme avec un paramètre additionel de la ligne de commande que vous récupérez dans la classe driver. Alternativement, vous pouvez dans le pire des cas réaliser un programme par tâche d'analyse à effectuer.**

- **Remarque: vous pouvez récupérer l'objet Configuration créé dans la classe driver depuis les classes map et reduce; et passer des valeurs par le biais de cet objet, ce qui peut vous aider fortement à diminuer les répétitions de code inutiles pour ce programme. Cherchez notamment comment obtenir l'objet Configuration depuis la variable context, et observez les méthodes get et set de cette classe Configuration.**

TP – Analyse de ventes

- **Votre programme doit être capable de répondre aux questions suivantes:**
  - **Obtenir le profit total obtenu par région du monde.**
  - **Obtenir le profit total obtenu par pays.**
  - **Obtenir le profit total obtenu par type d'item.**
  - **Pour chaque type d'item, obtenir deux valeurs:**
    - **La quantité de ventes en ligne.**
    - **La quantité de ventes hors ligne.**

    … et pour chacune d'entre elle, le profit total obtenu correspondant.