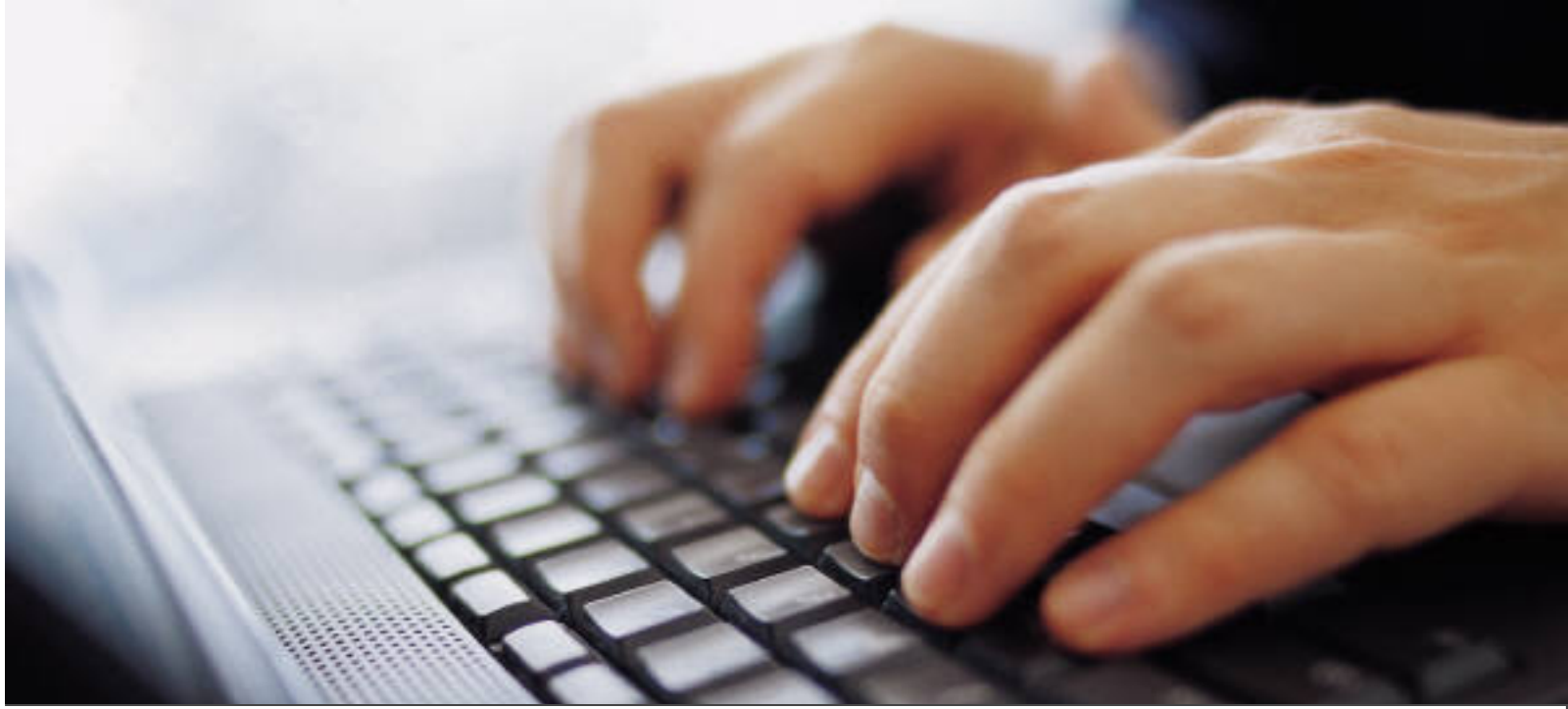
Hadoop / Big Data

Benjamin Renaut <renaut.benjamin@tokidev.fr>

MBDS

2019 – 2020



TP 2

Méthodologie Map/Reduce - programmation Hadoop plus avancée

Préparation du TP

- **La même machine virtuelle que précédemment est utilisée.**
- **Votre tâche consiste à implémenter un programme map/reduce exécutant l'algorithme de parcours de graphe vu précédemment.**
- **Pour rappel, vous pouvez lancer Hadoop avec la commande:**

```
start-hadoop
```

Parcours de graphes

- **Vous devez implémenter l'algorithme de parcours de graphes vu en cours.**

- **Il sera nécessaire pour cette tâche d'exécuter plusieurs fois le même programme map/reduce, en lui passant à chaque fois en données d'entrée les données de sortie de l'exécution précédente, et ce jusqu'à ce qu'une condition soit respectée.**

- **Les slides suivantes donnent quelques pistes concernant l'API hadoop pour des fonctions non vues en cours.**

- **Vous pouvez télécharger et extraire les données d'entrée via les commandes:**

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

Parcours de graphes

- **Notez bien le format d’entrée du fichier initial.**
- **Celui-ci respecte le format clef;valeur Hadoop standard: une clef, suivie d’une tabulation, suivie d’une valeur.**
- **Afin d’éviter que Hadoop ne passe chaque ligne d’entrée initiale comme un couple clef;valeur dont la clef est le numéro de ligne dans le fichier texte, il sera donc nécessaire de modifier le type d’InputFormat en conséquence.**
- **Le type a utilisé est un type standard Hadoop et a été présenté en cours.**

Parcours de graphes

- **A cette première étape, vous devez simplement réaliser l'implémentation de graphes sans implémenter d'InputFormat ou de types spécifiques.**

- **Vous devrez également lire les fichiers résultat sur HDFS dans la classe driver entre chaque exécution, pour déterminer si l'exécution doit continuer ou pas. Vous devez en conséquence rechercher dans la documentation Hadoop comment le faire depuis l'API.**

- **Pour vous aider dans votre recherche, regardez du coté de la classe:**

  `org.apache.hadoop.fs.FileSystem`

- **Si votre programme a été rédigé correctement, vous devriez à terme obtenir le même résultat d'exemple que celui vu en cours. Conservez les résultats d'exécution entre chaque lancement, ceci afin de vérifier le déroulement du programme.**

Objectifs – Partie 2

- **Vous devez désormais modifier votre programme pour qu'il utilise un type de valeur spécifique: GraphNodeWritable, que vous devez développer. Ce type qui stocker les différentes propriétés d'un nœud (couleur, voisins, etc.) et offrir des fonctions setter/getter « propres » pour leur utilisation au sein des classes map et reduce.**

- **Pour que cela fonctionne, vous devez utiliser un InputFormat et un OutputFormat spécifiques, qui vous sont fournis.**

- **Vous pouvez télécharger les adapteurs en question ici:**

  `http://cours.tokidev.fr/bigdata/tps/graph_io_formats.zip`

Objectifs – Partie 2

- **Votre classe GraphNodeWritable doit implémenter une méthode:**

  public String get_serialized()

  … qui doit renvoyer une chaine de caractère décrivant les propriétés du nœud (vous avez le choix du format); cette méthode est appelée par la classe OutputFormat spécifique fournie pour le TP.

  et doit proposer un constructeur:

  public GraphNodeWritable(String data)

  … qui doit prendre en paramètre une chaine du même format et remplir les champs en conséquence (déserialisation).

Remarques

- **Vous êtes fortement encouragés à consulter le code des InputFormat et OutputFormat spécifiques à GraphNodeWritable.**

- **Ceci vous permettra notamment de déterminer où vos deux méthodes à implémenter obligatoirement (get_serialized() et le constructeur de déserialisation) sont appelés. Vous pouvez également tenter de modifier le code fourni pour influer sur le format de sortie (nom de fichier ou format).**

- **Les quatre fichiers concernés sont:**
  `GraphInputFormat.java`
  `GraphOutputFormat.java`
  `GraphRecordWriter.java`
  `GraphRecordReader.java`

Remarques

- **Vous devez également préparer un fichier d'entrée; vous pouvez repartir du fichier graph_input.txt fourni en début de TP.**

- **Votre nouveau fichier doit être un CSV avec séparateur « ; » comportant deux champs par ligne: le premier étant la clef du nœud, le second la chaine sérialisée qui décrit le nœud dans votre format spécifique.**

- **Alternativement, un fichier vous est fourni: http://cours.tokidev.fr/bigdata/tps/graph_input.csv … vous devrez cependant dans ce cas impérativement respecter le format de sérialisation utilisé dans ce fichier dans votre classe GraphNodeWritable: chaque champs (voisins, couleur et profondeur) séparés par un caractère « | ».**

Bonus

- **Si vous avez fini en avance et que souhaitez un exercice plus difficile, vous pouvez tenter d'implémenter un algorithme de recherche en profondeur (DFS) en map/reduce.**

- **Cependant, soyez conscient que map/reduce est encore plus inadapté pour un depth first search, au point qu'il ne présente presque aucun intérét en terme de parallélisation et est encore plus « tordu » que le parcours en largeur.**