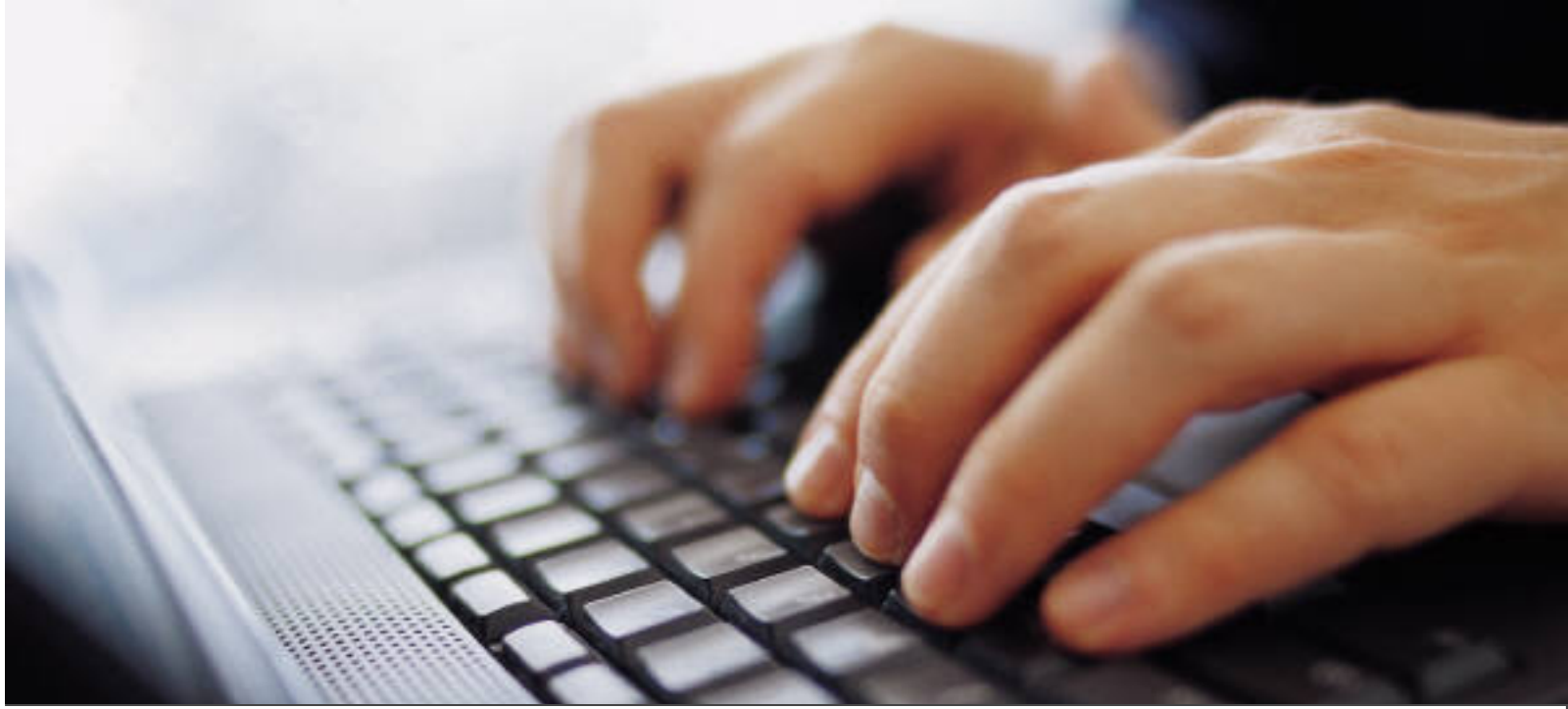
Hadoop / Big Data

Benjamin Renaut <renaut.benjamin@tokidev.fr>

MBDS

2019 – 2020



TP 2

Map/Reduce methodology – advanced Hadoop development

Setup

- **The same virtual machine that was used previously will be used here.**
- **Your task is to implement a map/reduce program that will perform the breadth-first search algorithm in a graph as described in the course material, and on the same example data.**
- **Reminder: you can start Hadoop using the command:**

```
start-hadoop
```

Breadth-first search

- **Implement the graph search algorithm described in the course material.**
- **For this task, and as described in the course, you will need to execute the same map/reduce program several times until a stopping condition (all nodes black), passing its previous output data as input data for the new run every time.**
- **The next slides give a few clue as to the functions you can research and use from the Hadoop API for this stopping condition.**
- **You can download the input data in the virtual machine using the command:**

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

Breadth-first search

- **Please note the format of the initial file.**
- **This format follows the Hadoop standard: one key;value tuple per line, with a tab between the key and the value.**
- **In order to avoid Hadoop passing you the data with the key being set as the line number from the initial text file, you will therefore need to use a specific InputFormat so that it correctly interprets the keys.**
- **The InputFormat to be used has been mentionned in the course material; you can also easily research it yourself.**

Breadth-first search

- **For this first part, you should simply implement the algorithm and the re-run logic.**

- **You *will* need to access HDFS directly from your driver main class, in order to check the output of the previous run to pinpoint wether or not your stopping conditions (all graph nodes black) has been reached.**

- **In order to help with this, lookup the documentation for the class:**

  `org.apache.hadoop.fs.FileSystem`

- **If your program works correctly, you should obtain the same final result as seen during the course. Please make sure to implement your program so that each intermediary step (the output of each run) is properly saved on HDFS.**

Part 2

- **You should now update your program so that it uses a specific Hadoop writable type: GraphNodeWritable, which you need to implement. This class should store the various graph node information (neighbours, colour, depth), and offer clean setters/getters to obtain and update those from the map and reduce classes.**

- **You will also need to use specific InputFormat and OutputFormat classes, which are provided.**

- **You can download the aforementioned classes here:**

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

Part 2

- **Your GraphNodeWritable class must implement the following method:**

  public String get_serialized()

  … which should return a serialized string describing the node (you pick the format); this method is called by the provided OutputFormat class.

  Your class also must have the following constructor:

  public GraphNodeWritable(String data)

  … taking a string with the same format and constructing a graph node object accordingly (unserialisation).

Part 2

- **It is highly advised to check out the code for the provided InputFormat and OutputFormat classes.**

- **This will among other things allow you to view where the methods your class has to implement (get_serialized() as well as the unserialisation constructor) are called. You can also attempt to change the provided code to change the output if you wish (for example, alter the file name or the format).**

- **The four provided files are as follows:**
  `GraphInputFormat.java`
  `GraphOutputFormat.java`
  `GraphRecordWriter.java`
  `GraphRecordReader.java`

Remarks

- **You also need to prepare a proper input file.**

- **Your new input file must be a CSV file, with a « ; » separator, and must have two fields on each line: the first being the graph node identifier, the second the serialized string that describes the graph node itself as per your chosen format.**

- **As an alternative, an example CSV file is also provided: http://cours.tokidev.fr/bigdata/tps/graph_input.csv**
  … if you decide to use this file, you will have to follow the same serialization format in your GraphNodeWritable class as is being used in it: every graph node field (neighbours, colour and depth) being separated by a « | » character.

Bonus

- **If you finished early and wish to attempt another task, you can try and implement a depth-first-search algorithm to go through the same graph (instead of the current breadth-first search approach).**

- **Be advised, however, that implementing DFS using a map/reduce approach can be quite counter-intuitive.**