# Big Data: distributed storage, processing and computing for data-intensive applications

## Lab 2: First steps with MapReduce

Dr. Maxime Guériau

maxime.gueriau@insa-rouen.fr

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

ITI4 – 2022

1. Setup
2. WordCount
3. Anagrams

# Objectives of this lab

1. Setup
   - □ To import and exploit a ready-to-use Hadoop VM
2. WordCount
   - □ To compile an existing MapReduce Java program
   - □ To run it through Hadoop
   - □ To understand how a MapReduce program works
3. Anagrams
   - □ To create and run your own MapReduce program
   - □ To design an algorithm within the MapReduce framework

# 1. Setup

1.1. Launch Oracle VM VirtualBox.

1.2. Import the VM named `BGD_VM_TM.ova` located in `/opt/ova`[1] in VirtualBox.

**Important:** Assign VM resources (virtual drive; with a new name) in `/tmp`!

1.3. Select the Machine/Settings/Network option and configure the network to 'host-only adapter/réseau privé hôte'. Choose the adapter named `vboxnet0`[2]. This will allow the guest machine to be part of a private subnetwork managed by the host machine.

1.4. Start the VM and log in (user:'bgd', password: 'password').

1.5. Check its IP address which should be in `192.168.0.0/24`:

```
ifconfig
```

1.6. Open a terminal on the host machine and check that it can ping the guest machine's IP address:

```
ping xxx.xxx.xxx.xxx
```

1.7. Establish a SSH connection to the guest machine:

```
ssh bgd@xxx.xxx.xxx.xxx
```

[1] Alternatively, on your own setup, you can download it from : `https://nuage.insa-rouen.fr/index.php/s/4pHeLATGPsKRDLa/download`

[2] On your own setup, you may need to create it first using the 'Host Network Manager' option of VirtualBox

# 2. WordCount

In the running SSH session, in a terminal opened on the host machine:

2.1. Check that you can find a file named 'book.txt' on the guest machine `/home/bgd/input` directory

2.2. Go to the `/home/bgd/java/wordcount` directory

2.3. Compile all `.java` files

```
javac WordcountMain.java WordcountMap.java WordcountReduce.java
```

2.4. Prepare the `.jar` file folders

```
mkdir -p bgd/hadoop/wordcount
mv Wordcount*.class bgd/hadoop/wordcount/
```

# 2. WordCount

In the running SSH session, in an opened terminal on the host machine:

2.5. Generate the `.jar` file

```
jar -cvf wordcount.jar -C . bgd
```

2.6. Go back to `/home/bgd` and run the following command line:

```
cd /home/bgd
hadoop jar java/wordcount/wordcount.jar bgd.hadoop.wordcount.WordcountMain file:///home/
    bgd/input/book.txt file:///home/bgd/output
```

2.7. Finally, display the results of the Hadoop MapReduce job using the following command line:

```
cat output/*
```

## A few remarks

- The output directory MUST be empty (or not yet created), you will need to empty it to run two successive jobs in the same directory.

  ```
  rm -R output/
  ```

- The procedure described in this section 2 can be reused to build, compile and run your own MapReduce jobs.
- It is also possible to compile your code and generate a `.jar` file on the host machine and then copy and execute it on the guest machine.

# 2. WordCount

2.8. Go back to the `/home/bgd/java/wordcount` directory

2.9. Go through all `.java` files and try to answer the following questions:

- What are the (key,value) pairs used in this program?
- Why is there two of them?
- What is the difference between (KEYIN,VALUEIN) and (KEYOUT,VALUEOUT)?
- What are the main constraints of the MapReduce framework?
- Are we really benefiting from the distributed computing capabilites of Hadoop?

# 3. Anagrams

3.1. On the guest machine, have a look at the file named 'english_words.txt' and located in `/home/bgd/input`

3.2. On the host (or guest), copy all `.java` files from the Wordcount program you used in the previous exercise.

3.3. Starting from this code, build a new MapReduce program that finds all anagrams in a given `.txt`. The expected output should look like this:

```
[yawn, awny, wany]
[any, nay, yan]
[pottaro, portato, potator, taproot]
[potato, pattoo, topato]
[proart, parrot, raport, raptor]
```

3.4. Copy the results on the host machine; check if your program is working as expected.

## Tips and remarks

- An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once (ex: 'binary' and 'brainy').
- Before implementing, think wisely about your input/output (key,value) pairs
- You can write a bash (`.sh`) script for saving time (compilation on host; copy to guest; and execution on guest)

# 3. Anagrams

## Need more help?

- The key to create a MapReduce algorithm is to think in terms of desired (key,value) pairs.
- Here, you need to find a function that produces the same key for all words (values) that are anagrams. This way, they will be forwarded to the same reducer.
- A nice way to do it is to sort all the characters in all the words: all anagrams will have the same key! Example:

  ```
  listen => eilnst
  silent => eilnst
  ```

- You can reuse the following code in your MapReduce program:

  ```
  import java.util.Arrays;

  public static String sortCharacters(String input) {
    char[] cs = input.toCharArray();
    Arrays.sort(cs);
    return new String(cs);
  }
  ```