Project 2: Hadoop MapReduce "from scratch" on Java

I Sequential Word Counter

I.1 Basic Functionality

Count the number of occurrence of words in a file by using only one processor. You may choose a data structure among **List, HashMap, HashSet, LinkedHashSet** or another. Justifier yourself in the report.

Test your program with a file *input.txt* with the content:

Deer Beer River

Car Car River

Deer Car Beer

Result:

Deer 2

Beer 2

River 2

Car 3

I.2 Sortage

Sort your result first by the number of occurrence, then by the name of the word.

Result:

Car 3

Beer 2

Deer 2

River 2

I.3 Test

Test your program with a larger file. You can find a file from:

https://github.com/legifrance/Les-codes-en-vigueur

For example, in *deontologie_police_nationale.txt*, what are the 5 most popular words?

If you want to find a larger file, http://commoncrawl.org/the-data/get-started/ is a good place. The Common Crawl corpus contains petabytes of data collected over the last 7 years. It contains raw web page data, extracted metadata and text extractions.

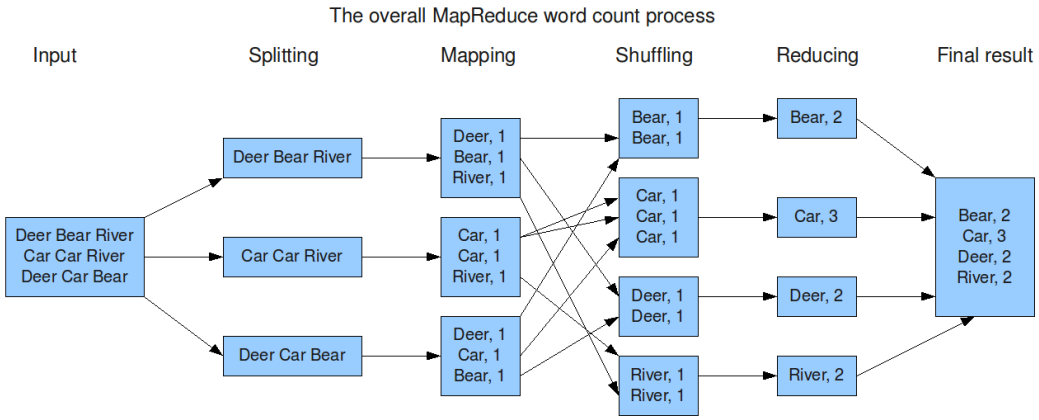
II MapReduce

In this part, you have some options to simulate the test environment.

1. You may use your friends' computers as remote computation resources.
2. You may create several virtual machines on your laptop.
3. You may use tools like Docker to create containers.
4. Other methods.

You can choose whatever method that suits your group. What we need is several accessible nodes that eventually become our distributed system. Don't forget to test the connection before going to the next step. For your cluster, it would be easier if you establish the connection without the password among each other, you may check this if you need: https://www.shellhacks.com/disable-ssh-host-key-checking/

You may also learn some Linux commands: **ls, mkdir, cat, cp, ssh, scp**…



II.1 Remote Execution

Create a SLAVE program which calculate 1+1 and print the result. Export it as a Jar file.

Create a folder in your distant computer */tmp/<your name>/*. You may do all your test in this folder.

Transfer your Jar file to the remote computer.

Execute the program remotely. You might see nothing from the command line so store the result in a file and check if it works.

II.2 Master Program

Create a program MASTER which will launch another program by command line.

You may consult this:

http://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html

For example, to list the files in a folder:

ProcessBuilder pb = new ProcessBuilder("ls", "-al", "/tmp");

Try to test the MASTER program by launching the SLAVE program.

What if we add *sleep(100000)* in SLAVE program before doing the actual calculation? Try also *sleep(10000)*, *sleep(1000)*.

You may also manage the standard output stream and error stream.

One possible solution may be as follows:

1. Redirect the output stream with the method *pb.inheritIO()*
2. Wait until the end of the process by

boolean b = p.waitFor(3, TimeUnit.SECONDS);

You can also use 2 threads respectively for the standard output stream and the error stream. This solution might be a little bit difficult.

II.3 Deploy Program

Create a file including the hostnames or ip addresses of the computers you would use for the distributed system.

Create a new program named DEPLOY which will read the file and test the connection. You might use the command *nmap*. This helps to verify whether the computers are not turned off or other situations.

Modify your DEPLOY program and now create your test folder with your name in the distant available computers. Transfer your SLAVE program with *scp*.

After the DEPLOY, your MASTER should be able to execute SLAVE on a distant computer. How do you make sure the *scp* command happens after *mkdir* command?

Split the original file into 3.

S0.txt: Deer Beer River

S1.txt: Car Car River

S2.txt: Deer Car Beer

Modify your DEPLOY program to transfer *Sx.txt* to */tmp/<your name>/splits/* on distant computers. You might need to first create the folder.

II.4 SLAVE program

Modify your SLAVE program which takes arguments as the mode of functionalities. 0 represents the mode of Map. It takes a file *Sx.txt* in *splits* generate a file *UMx.txt* in *maps*.

Test the Map:

cd /tmp/<your name>/

java -jar slave.jar 0 /tmp/<your name>/splits/S0.txt

The file */tmp/<your name>/maps/UM0.txt* should be like:

Dear 1

Beer 1

River 1

What would be the result of *UM1.txt*?

Modify your SLAVE program to print keys:

Dear

Beer

River

Modify your MASTER program. Launch the SLAVE remotely. MASTER can receive keys by the standard output stream. Wait until all the Maps are terminated and print "Phase Map terminated".

Add mode 1 for SLAVE program (phase Shuffle) which takes several *UMx.txt* as input and generate *SMx.txt*. The number of *SMx.txt* will indicate the number of key.

Test with *UM1.txt* and *UM2.txt:*

cd tmp/<votre nom d'utilisateur>/

java -jar slave.jar 1 Car /tmp/<your name>/maps/SM1.txt /tmp/<your name>/maps/UM1.txt /tmp/<your name>/maps/UM2.txt

The file /tmp/<your name>/maps/SM1.txt should be like:

Car 1

Car 1

Car 1

Modify your SLAVE program and add mode 2 (Phase Reduce) which takes *SMx.txt* as input and generate *RMx.txt*.

Test your implementation:

cd tmp/<your name>/

java -jar slave.jar 2 Car /tmp/<your name>/maps/SM1.txt /tmp/<your name>/reduces/RM1.txt

The file */tmp/<your name>/reduces/RM1.txt* should be like:

Car 3

II.5 Finally

Modify your MASTER program and implements Phase Shuffle, Reduce and gather final result of the sorting. Make sure at the end of each phase, print something to indicate.

III Experiment

III.1 Record the Time

You may compare the time consumed both by a trivial method and the MapReduce method. You can also record each phase of the MapReduce and find out which phase takes the most.

III.2 Test on Large Files

You may test the MapReduce on large files and make some reflection.

If the result is not ideal, think about what could be optimized.