CentraleSupélec 2017-2018 **Big Data Platforms : Lab 2**

MapReduce - Hadoop

The purpose of this tutorial is to get you started with MapReduce and Hadoop (HDFS and MapReduce execution), two core elements of the Big Data stack. You will work with JAVA and the ECLIPSE Integrated Development Environment (not mandatory)

Learning outcomes :

— Getting familiar with the Hadoop framework : HDFS, MapReduce, Yarn.
— Writing and running some data processing problems with MapReduce and Hadoop.
— Running a distributed computing an a cloud.

Exercice 1 Apache Hadoop installation

Your first work is to install Hadoop on your machine, either **directly** or with a **virtual machine** according to your OS (Linux is better for Hadoop).

Hadoop can be run in three modes :

— **Standalone (or local mode)** : in this mode, no daemons are used (everything runs in a single JVM). Hadoop uses the local file system as a substitute for HDFS file system. In this configuration, we considerer an execution environment with 1 mapper and 1 reducer. This mode is often used for running MapReduce programs during development.
— **Pseudo-distributed mode** : this mode mimics the behavior of a cluster but all the daemons run on a single local machine (different JVMs are locally executed). This mode can accept multiple mappers and reducers and uses HDFS.
— **Fully-distributed mode** : hadoop on a real cluster.

In this tutorial, you will not use a real hadoop cluster and as a consequence, you will work only in local mode (standalone or pseudo-distributed).

Question 1.1 First case : installing Hadoop directly

The manual installation of Hadoop is quite simple. According to your OS, you can refer to these different tutorials :

— **Windows** : `https://wiki.apache.org/hadoop/Hadoop2OnWindows`
— **Linux** : `https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-common/SingleCluster.html`
— **Mac OS** : `https://dtflaneur.wordpress.com/2015/10/02/installing-hadoop-on-mac-osx-el-capitan/`

Most of the time, installing Hadoop is done with the following steps :

1. System update and required software installations.
   — Java must be installed (at least v7). You can verified that you have a correct version of java using the following command :
   `$ java -version`
   — ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.
2. Building of a group and a specific user for Hadoop (not mandatory for your usage).

```
$ addgroup hadoop
$ adduser --ingroup hadoop hadoopuser
$ adduser hadoopuser
```

3. Configuration of ssh to enable to access to localhost for the hadoopuser

```
$ ssh-keygen -t rsa -P ""
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ ssh localhost
```

4. The Hadoop installation
   (a) Downloading Hadoop from a mirror site : `https://www.apache.org/dyn/closer.cgi/hadoop/common/`
   (b) Untar(zip) and installation of the archive in a chosen repertory : MYHADOOPREPERTORY
   (c) Go to this repertory

   ```
   $ cd MYHADOOPREPERTORY
   ```

   (d) Specify the location of JAVA into the file `etc/hadoop/hadoop-env.sh` . This location can be obtained with the following command :

   ```
   /usr/libexec/java_home
   ```

   And then :

   ```
   export JAVA_HOME={JAVA_HOME}
   ```

   (e) Try the following command

   ```
   $ ./bin/hadoop
   ```

5. Create an environment variable that points to your Hadoop installation directory : MYHADOOPREPERTORY

```
$ export HADOOP_HOME = MYHADOOPREPERTORY_PATH
$ export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

6. Check your Hadoop installation by running :

```
$ hadoop version
```

7. Hadoop configuration through the editing of different configurations files.
   Each component in Hadoop is configured using an XML file. Common properties are in the file `core-site.xml` and properties pertaining to HDFS, MapReduce and YARN go into the appropriately named files : `hdfs-site.xml`, `mapred-site.xml` and `yarn-site.xml`

   (a) Configuring Hadoop in standalone mode by editing the file `etc/hadoop/core-site.xml`

   ```
   <configuration>
       <property>
           <name>fs.defaultFS</name>
           <value>hdfs://localhost:9000</value>
       </property>
   </configuration>
   ```

   Here, the name of the file system has been specified : all the files and repertories in HDFS will be prefixed by `hdfs://localhost:9000`. In the standalone mode by default the value of this property is `file:///`

   (b) Configuration of the parameters of the HDFS file system by editing the file `etc/hadoop/hdfs-site.xml`

   ```
   <configuration>
       <property>
           <name>dfs.replication</name>
           <value>1</value>
       </property>
   </configuration>
   ```

(c) Specification of the parameters of MapReduce by editing the file `etc/hadoop/mapred-site.xml`

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Here, we specify that Yarn is used as the implementation of MapReduce.

(d) At last, we can parameter Yarn by editing the file `etc/hadoop/yarn-site.xml`

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

Here, we precise that an operation SHUFFLE will be used.

8. At this step, Hadoop is correctly installed and configured, we will just have to format the local HDFS system file and to start Hadoop

```
$ hdfs namenode -format
$ start-dfs.sh
$ start-yarn.sh
```

Question 1.2 Second case : Using a virtual machine environment

An alternative (recommanded for not linux OS) is to use a virtual machine environment and a package distribution of Hadoop. You also have to do this work in order to use Hadoop in pseudo-distributed mode easily. For this case, we will follow the following Hadoop tutorial : `http://web.stanford.edu/class/cs246/homeworks/hw0/tutorialv3.pdf`. In the case, you will use the Cloudera's distribution which is an integrated Apache Hadoop-based stack.

1. Download and install *VirtualBox* on your machine : `https://www.virtualbox.org/`.
2. Download the *Cloudera Quickstart VM* : `https://www.cloudera.com/downloads/quickstart_vms/5-12.html`
3. Uncompress the VM archive.
4. Start *VirtualBox* and click *Import Appliance* in the *File* dropdown menu. Click the folder icon beside the location field. Browse to the uncompressed archive folder, select the .ovf file, and click to the *Open* button. Click the *Continue* button. Click the *Import* button.
5. Your virtual machine shoud now appear in the left column. Select it and click on *Start* to lauch it.
6. To verify that the VM is running and to access it, open a browser to the URL : `http://localhost:8088`. You should see the ressource manager UI. The VM uses port forwarding for the common Hadoop ports, so when the VM is running, those ports on localhost will redirect to the VM.

Exercice 2 Exploring Hadoop Distributed File System (HDFS)

In this part of the tutorial, you will learn about HDFS (how it works, best practices...). HDFS allows user data to be organized in the form of files and directories. At the end of this part, you will be able to :

— Verify HDFS is running and check the health of an HDFS file system.
— Get data in and out of HDFS using both the HDFS commands and the HDFS java API
— Understand strategies for streaming data into HDFS.

Memo 1.

HDFS (Hadoop Distributed FileSystem) is a filesystem that has been designed for storing very large files with streaming data access patterns and running on clusters of commodity hardware. Its main concepts are :

— Files in HDFS are broken into **block-sized chunks** (128 MB by default) which are stored as independent units. The size of HDFS blocks is large in order to minimize the cost of seeks. It has several properties :
  — A file can be larger than any single disk in the network.
  — It simplifies the storage subsystem (fixed size, blocks are just chunks of data, metadata are handled separately)
  — Blocks fit well with replication (fault tolerance and availability).
  — **Namenodes and Datanodes**
  An HDFS cluster has two types of nodes : a **namenode** (the master) and a number of **datanodes** (workers).
    — The namenode manages the filesystem namespace. It maintains the filesystems tree and the metadata for all the files and directories in the tree. It also knows the datanodes an which all the blocks for a given file are located.
    — Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to. They also report back to the namenode periodically.
    — **Block caching**
    Frequently accessed file blocks can be explicitly cached in the datanode's memory (`block cache`).

Question 2.1 Main HDFS commands

In this part, we will overview the main and most frequently used HDFS commands. A more comprehensive list of file system commands and options can be found on the Hadoop project website : `http://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-common/FileSystemShell.html`

Memo 2.

The base scheme for HDFS commands is the following :
`hdfs dfs -cmd [args]`
We can list all the available commands with :
`hdfs dfs`

Try and understand the following commands from the shell of your Hadoop client :

1. `hdfs dfs` (same as `hadoop fs`)

2. `hdfs dfs -ls` : this command displays the files on the Hadoop file system. (Try the linux command `ls -l`) to see the difference)
3. `hdfs dfs -ls /`
4. `hdfs dfs -mkdir -p /user/myusername` : create your home directory in HDFS which is in that case `/user/myusername`
5. `hdfs dfs -mkdir fichiersLab2` : to create a directory for this lab
6. Build a file that contains the word Hello and named *Hello.txt* and add it to HDFS with the following command :
   `hdfs dfs -put Hello.txt`
   Where is this file in HDFS ?
7. Try `hdfs dfs -ls -R`.
8. `hdfs dfs -cat Hello.txt` : to view the content of the file on HDFS
9. `hdfs dfs -cp Hello.txt Salut.txt` : to copy the file. Verify using `hdfs dfs -ls`.
10. `hdfs dfs -rm Salut.txt` : to remove a file. Verify using `hdfs dfs -ls`.
11. Try `hdfs dfs -rm Hello.txt` and `hdfs dfs -copyfromLocal Hello.txt` . What's happened ?
12. `hdfs dfs -get Hello.txt Salut.txt` : download the file from HDFS back to the client.
13. `hdfs fsck /` : check the entire HDFS filesystem for inconsistencies/problems

Question 2.2 HDFS Best Practices

A common practice for HDFS storage is to place files of a common data set into the same folder. The following steps are common in HDFS :

1. Start by making a folder for the dataset.
2. Upload all the targeted files into this folder.
3. Verify the files are there within the `hdfs dfs -ls` command.
4. Use the command `hdfs dfs -cat folder/*` to view all the content of a folder as a single file.
5. `hdfs dfs -getmerge folder/* result.txt` to download all the files of a folder and return them as a single file. Why this command useful ?

Question 2.3 HDFS Block Storage

HDFS splits file into blocks and those blocks are then distributed to datanodes managed by HDFS. For a disk, a block is the minimum amount of data that can be read or write. In HDFS, the size of a block is by default 128MB. Files in HDFS are broken into block-sized chunks, which are stored as independent units.

The command `hdfs fsck / -files -blocks` lists all the blocks that make up each file in the filesystem. You can also specify the size of a block size when you upload file on HDFS. For instant, try to put the file Categorie.csv (which in on the drive) using the following commands :

— `hdfs dfs -D dfs.blocksize=500 -put Categorie.csv folder/`
— `hdfs dfs -D dfs.blocksize=1200500 -put Categorie.csv folder/`
— `hdfs dfs -D dfs.blocksize=1048576 -put Categorie.csv folder/`

Question 2.4 HDFS Health check

The command `hdfs dfsadmin -report` gives you a health check report of the HDFS file system.

Question 2.5 On your own !

1. You will work with the meteorological data available here : `https://www1.ncdc.noaa.gov/pub/data/noaa/`. Try to understand these data. You will see that datafiles are organized by date and weather station.
   You will have to build a script (.sh) that will download and store on HDFS the data of four stations of your choice for the year 2015, 2016 and 2017.

Question 2.6 Working with the HDFS API

Memo 3.

Hadoop provides a complete API to manage HDFS files. It has two main classes :
- `FileSystem` : which represents the file tree (*file system*). It enables to copy local files to HDFS (respectively from HDFS to your local file system), to rename, create ad remove files and directories.
  `https://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FileSystem.html`
- `FileStatus` : which manages all the informations about files and directories :
  - size with `getLen()`.
  - nature with `isDirectory()` and `isFile()`.
    `https://hadoop.apache.org/docs/r2.7.3/api/org/apache/hadoop/fs/FileStatus.html`

  These two classes need to know the configuration of the HDFS cluster (`Configuration`).
  Paths to files are managed with the class `Path`.

Hadoop reference guide book : pages 56-78

In this part, you will use the HDFS JAVA API and you will use the Eclipse IDE. You can use the template project given in the course website. For this, you will have to :

1. Download the file `TPIntroHadoopHDFS.zip` on your computer.
2. In Eclipse, click on `File > Import ...`
3. In the General category, select `Existing Projects into Workspace`
4. Select `select archive file` and find the targeted archive.
5. Click on `Finish`.

Question 2.6 .1 A first simple program

In a first program, you will write a program that counts the lines of the file `arbres.csv` that you have to put on HDFS. For this, you have to build a new Eclipse project. For this, you have the file `CountLineFile.java` that displays each line of a given file. You just have to adapt the code source.
- To run your Hadoop program in standalone mode, you just have to run it as a java application (except if you have give the name of the file as an argument of the main method).
- To run your project in pseudo-distributed mode, you have to transform it into a compiled packet. In Eclipse, you have to export it into a JAR file on the local file system.
  - Right-click on the project and select `Export`.
  - In the pop-up dialog, expand the `Java` node and select JAR file.
  - Enter a path in the `JAR file` field and click `Finish`.
  - Open a terminal and run the following command :

```
hadoop jar path/to/file.jar [Main class] [input path] [output path]
```

Question 2.7 Displaying the content of a CSV file (to upload)

The objective of this exercice is to display the year and the height of each tree of the file `arbres.csv`. This file describes the amazing trees of Paris (from `http://opendata.paris.fr`. For this, you will have to build a new Eclipse project named `YearHeightTree` by copying/pasting the previous one and refactoring (renaming) it. In this project, we will have to build a class named `Tree` used to represent one of the line of the file (that corresponds to one tree). Try to build it in such a way that it will not be instanciated.

You will have to upload this project on a git repository..

Question 2.8 Displaying the content of a compact file (to upload)

We will work with the file `isd-history.txt` of the NOAA. This file is available here : `https://www1.ncdc.noaa.gov/pub/data/noaa/isd-history.txt`. Put it on HDFS and then use the command `hdfs -dfs - tail file` to see this last lines. Your job is to write a program using the HDFS API that will display the USAF code, the name, the country (FIPS country ID) and the elevation of each station.

Some important points :

— The first 22 lines of the file have to be ignored (they do not contain data).
— The name of each station begins at the character 13 of a line and its size is 29 characters long.
— The FIPS begins at the character 43 of a line and its size is 2 characters long.
— The altitude begins at the character 74 of a line and its size is 7 characters long.

You also have to upload this project on your git repository.

Exercice 3 Map Reduce : Wordcount

Memo 4.

MapReduce is a programming model for data processing. It works by breaking the processing into two phases : the **map** phase and the **reduce** phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. To write a MapReduce job with the JAVA API of Hadoop, you have to :

— Extend the `Mapper` class of the Hadoop API and override the abstract`map` method.
— Extend the `Reducer` class of the Hadoop API and override the abstract`reduce` method.
— Build a general and main class (often named `Driver`) that builds a job making some references to the two previous classes. This main class extends the `Configured` class of the Haddop API and implements the `Tool` interface.

In this section, you will implement with Hadoop the Hello world of MapReduce : WordCount. You will work on the text of Anna Karenina of Tolstoy available here : `http://www.textfiles.com/etext/FICTION/anna_karenina`.

Question 3.1 Simple Wordcount

If you have done the section 2.5 of the Hadoop tutorial : `http://web.stanford.edu/class/cs246/homeworks/hw0/tutorialv3.pdf` then you can skip this part.

Memo 5.

Hadoop uses the `Writable` interface based classes as the data types for the MapReduce computations. It defines how Hadoop should serialize (translating data structures or object state into a format that can be stored and transmitted, RPC) and de-serialize the values when transmitting and storing the data. Available datatypes are :

- `Text` : any UTF8 string
- `BooleanWritable` : a boolean
- `IntWritable` : 32-bits integer
- `LongWritable` : 64-bits integer
- `FloatWritable` : 32-bits IEEE float
- `DoubleWritable` : 64-bits IEEE double

Memo 6.

Hadoop supports processing of many different formats and types of data through `InputFormat`. The `InputFormat` of a Hadoop MapReduce computation generates the key-value pair inputs for the mappers by parsing the input data. It also performs the splitting of the input data into logical partitions, essentialy determining the number of map tasks of a MapReduce computation (a map task for each logical data partition).

- `TextInputFormat` : used for plain text files. It generates a key-value record for each line of the input text files. It is the default `InputFormat` of Hadoop.
- `NLineInputFormat` : used for plain text files. It splits the input files into logical splits of fixed number of lines.
- `SequenceFileInputFormat` : ...

Memo 7.

The output of your MapReduce job is often consumed by other applications and as a consequence, it is important to store its results into a suitable format. Hadoop provides the `OutputFormat` interface to define the data storage format, data storage location and the organization of the output data of your MapReduce job.

If no, you can use the template eclipse project given on the website of the course (TPIntroHadoopMapReduce.zip). The main points are :

- Build a new project by copying the template project.
- Create a new package called `cs.Lab2.WordCount`.
- Create three classes in that package : `WordCountMapper` that extends the class `Mapper` of the Hadoop java API, `WordCountReducer` that extends the class `Reducer` of the Hadoop java API and `WordCountDriver` that extends the class `Configured` and that implements the interface `Tool` of the Hadoop java API. Write the code of these classes in order to count the number of occurences of each word in a given document (as seen in course).
- Compile your code either using the menu `Run as -> Run Configurations` menu or directly using `javac`.

```
$ export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)
$ javac -classpath $HADOOP_CLASSPATH WordCount*.java
```

- Then export your code as a .jar either using the `Export` menu or directly using :

```
$ mkdir -p cs/Lab2/wordcount
$ mv *.class cs/Lab2/wordcount
$ jar -cvf cs_Lab2_wordcount.jar -C . cs
```
— Use Hadoop to run WordCount on your file.
```
$ hadoop jar cs_Lab2_wordcount.jar cs.Lab2.wordcount.WordCountDriver
 /input/anna-karenina.txt /results
```
— Run the command `hadoop fs -ls /results`
— Use the interface at `http://localhost:8088` to see the logs of the job

Question 3.2 Wordcount with combiners

Memo 8.

Combiner is a semi-reducer in mapreduce. This is an optional class which can be specified in mapreduce driver class to process the output of map tasks before submitting it to reducer tasks. The main function of a Combiner is to summarize the output from Map class so that the stress of data processing from reducer can be managed and network congestion can be handled.
Unlike mapper and reducer, combiner do not has any predefined interface. It needs to implement the reducer interface and overwrite `reduce(` method.

Modify the previous program to :
— Add a combiner with :

```
job.setCombinerClass(...)
```
— Modify your code in precising the number of reduce tasks.
```
job.setNumReducetasks(int tasks)
```

Question 3.3 Wordcount with in-mapper combiners

As seen in the course, the in-mapper combining design pattern may result in a more efficient algorithm implementation by reducing both the number and size of key-value pairs that need to be shuffled from the mappers to the reducers.

1. As seen in the course, use an associative array (i.e. a Map in Java) to store the words and their associated frequency in the `map` function. After we have counted all the words in the incoming text, then we emit each word and its associated frequency. Modify your word count mapper to implement this.
2. Instead of using an associative array per key-value input, use the in-mapper approach that use an associative array per mapper. Modify your code.

Exercice 4 Computing the mean

In this section, you will use the NCDC weather dataset. Your aim is to calculate the average temperature for each month in the year 1954. The average algorithms for the combiner and the in-mapper combining option are given below in pseudo code. Try and compare the different approaches.

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)
1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

FIGURE 1 – Basic Map Reduce for the mean - Source : Jimmy Lin and Chris Dyer book

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)
1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))          ▷ Separate sum and count
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Figure 3.5: Pseudo-code for an incorrect first attempt at introducing combiners to compute the mean of values associated with each key. The mismatch between combiner input and output key-value types violates the MapReduce programming model.

FIGURE 2 – Basic Map Reduce with incorrect combiner for the mean - Source : Jimmy Lin and Chris Dyer book

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))
1: class Combiner
2:     method Combine(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Figure 3.6: Pseudo-code for a MapReduce algorithm that computes the mean of values associated with each key. This algorithm correctly takes advantage of combiners.

FIGURE 3 – Basic Map Reduce with a correct combiner for the mean - Source : Jimmy Lin and Chris Dyer book

```
1: class Mapper
2:     method Initialize
3:         S ← new AssociativeArray
4:         C ← new AssociativeArray
5:     method Map(string t, integer r)
6:         S{t} ← S{t} + r
7:         C{t} ← C{t} + 1
8:     method Close
9:         for all term t ∈ S do
10:            Emit(term t, pair (S{t}, C{t}))
```

Figure 3.7: Pseudo-code for a MapReduce algorithm that computes the mean of values associated with each key, illustrating the in-mapper combining design pattern. Only the mapper is shown here; the reducer is the same as in Figure 3.6

FIGURE 4 – Basic Map Reduce with a in-mapper combiner for the mean - Source : Jimmy Lin and Chris Dyer book

Exercice 5 Your own MapReduce programs

This part of the work will have to be upload on a git repository. For each MapReduce problem, you have to submit the following files :

— the source code of your program in JAVA.
— A brief document that gives the answer to the question asked in the problem description.
— A screen-shot image of your EMR Job Flows console that shows your program's COMPLETED state as well as the elapsed time and your AWS account name.

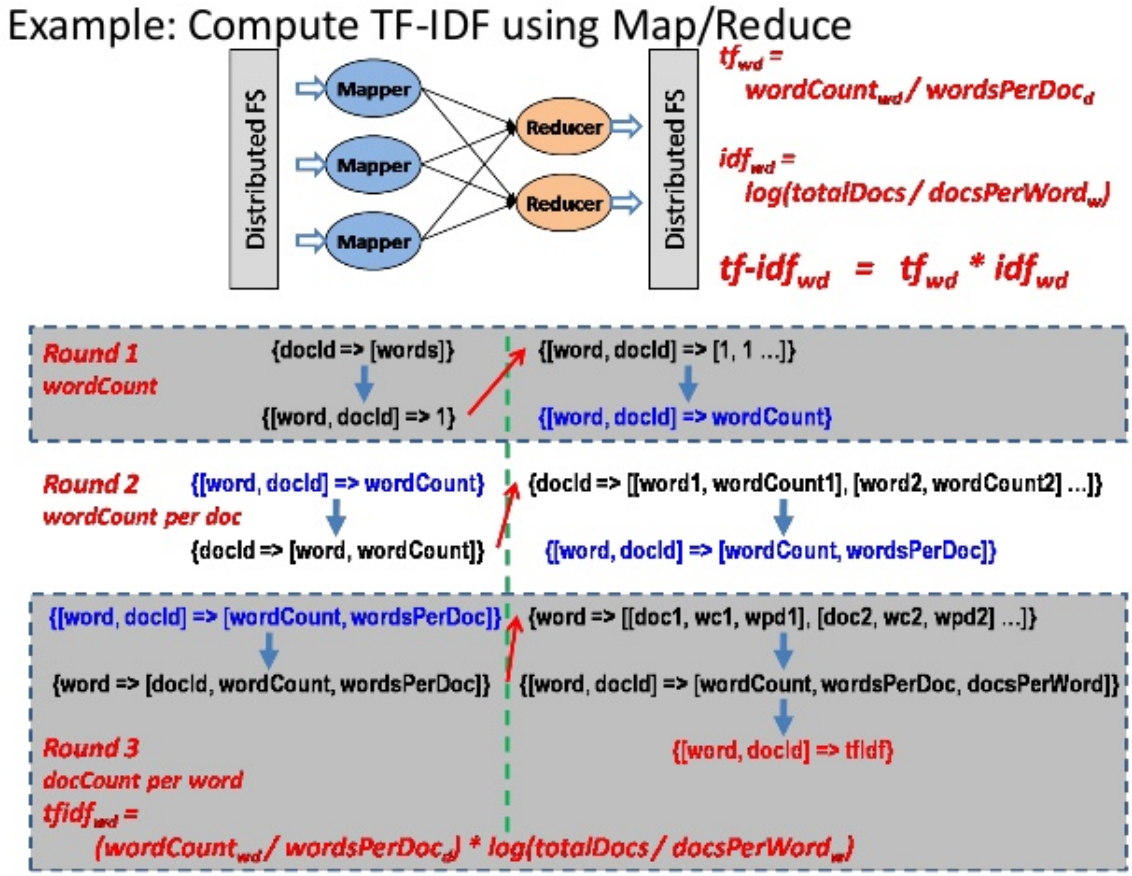
Question 5.1 Problem 1 : TF-IDF (to upload)

Your goal is to calculate Term Frequency-Inverse Document Frequency (TF-IDF) of a set of documents using MapReduce. In particular, you will use the following documents that you will add to HDFS

```
hadoop fs -mkdir input
 wget http://www.textfiles.com/etext/FICTION/defoe-robinson-103.txt
hadoop fs -copyFromLocal defoe-robinson-103.txt input
wget http://www.textfiles.com/etext/FICTION/callwild
hadoop fs -copyFromLocal callwild input
```

Some links :

— `https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html`
— `http://www.dcs.bbk.ac.uk/~dell/teaching/cc/book/ditp/ditp_ch4.pdf`



Which 20 words have the highest tf-idf scores in these documents ? List them in descending order.

Question 5.2 Problem 2 : Page Rank (to upload)

Write a MapReduce program to calculate the PageRank score (with damping factor 0.85) for each user in the Epinions who-trust-whom online social network : `https://snap.stanford.edu/data/soc-Epinions1.html`. Which 10 users have the highest PageRank scores in this social network ? List them in descending order.

Some links for this problem :

— `http://www.dcs.bbk.ac.uk/~dell/teaching/cc/dell_cc_07.pdf`
— `http://www.dcs.bbk.ac.uk/~dell/teaching/cc/book/ditp/ditp_ch5.pdf`
— `http://infolab.stanford.edu/~ullman/mmds/ch5.pdf`
— `http://www.dcs.bbk.ac.uk/~dell/teaching/cc/paper/mlg10/lin_mr_graph.pdf`

Question 5.3 Problem 3 : The trees of Paris (to upload)

The objective of this problem is to compute some information on the trees of Paris using the file `arbres.csv`. Write some MapReduce programs that :

— Compute the number of trees by type.
— Compute the height of the highest tree of each type.
— Computer the borough of the oldest tree in Paris (**optional**).