Lab 1

This introductory lab is composed of three tasks. Your final objective is to run your first Hadoop application. For this goal, you must learn how to compile the source code and produce a jar, connect to a remote machine, transfer files using FTP and submit an application on a cluster.
Please note:

- at LABINF, you have all the software you need already installed in **Linux**
- you can see **the architecture of the BigData@Polito** environment in the slides on the course web page; you need to understand the difference among
  - your local machine (LABINF PC),
  - the bigdatalab.polito.it remote gateway,
  - the Hadoop cluster with its own HDFS distributed file system

http://dbdmg.polito.it/wordpress/wp-content/uploads/2018/03/00_Cluster_BigData_2x.pdf

1. Compiling using an IDE (Eclipse)

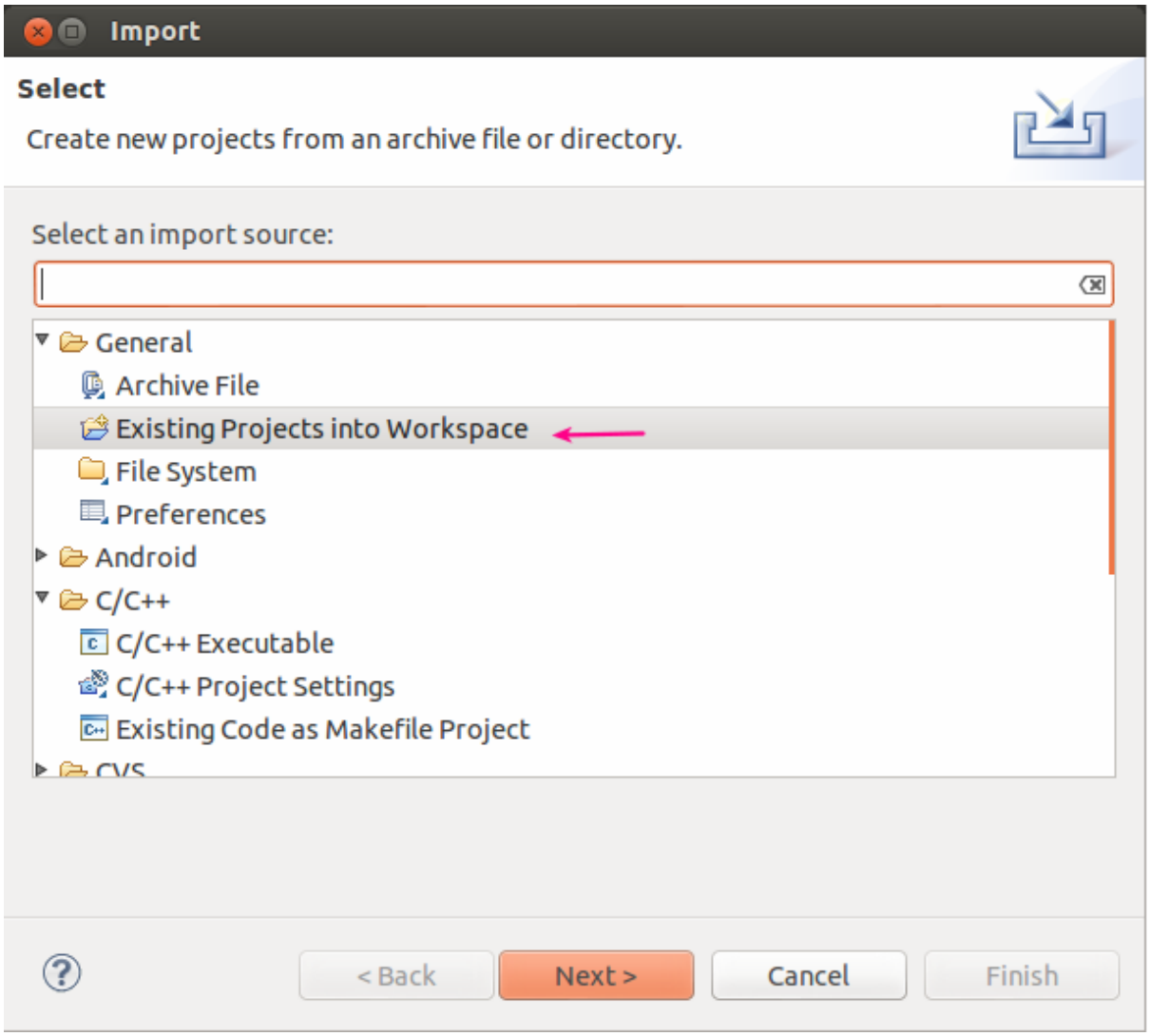
In this task, we will compile the source code of a simple Hadoop application. The easiest way would be using Maven, but here we will guide you through the manual creation of a jar instead. Remember that, if you don't use Maven, you will need to manually include the libraries every time you create a new project.
These instructions explain how to import manually the libraries (that we provide to you inside the lib/ folder of the zip). If you do not have enough free space in your home, unzip the libraries to a folder inside /tmp/, but remind that these files will be deleted at your logoff.

This first application is a simple word count.
You can use the solution available on the course web page: Lab1_BigData_with_libraries.zip

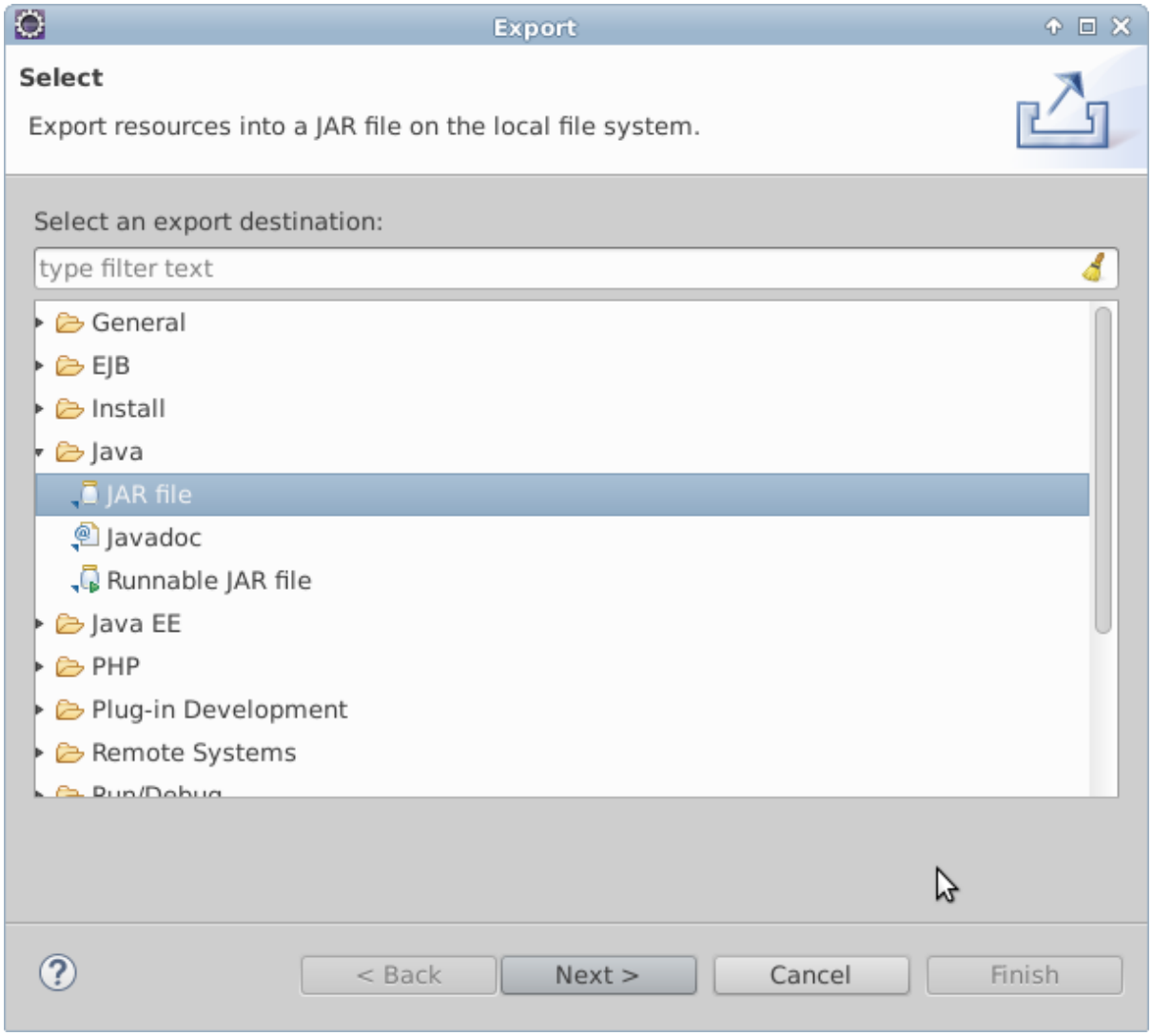
- ➔ direct link: http://dbdmg.polito.it/wordpress/wp-content/uploads/20186/03/Lab1_BigData_with_libraries.zip
- ➔ please note that Eclipse works on your local machine e generates a JAR file on your local machine (i.e., the LABINF PC)

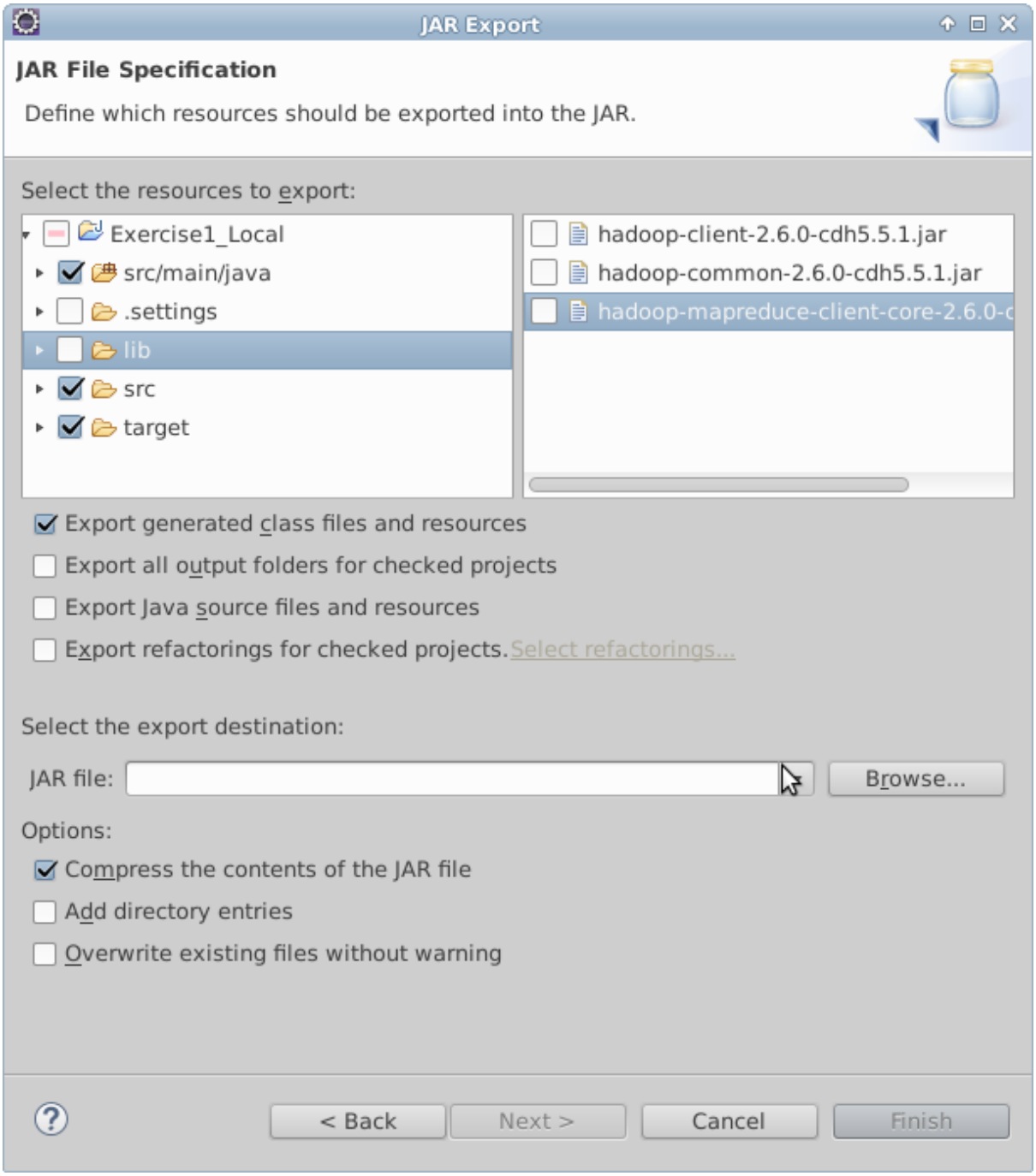
1. Open Eclipse

2. Import the project (File -> Import… | General -> Existing project... | choose the folder where you extracted your file)



3. Have a look at the source files and the structure of the project. Where is the mapper? Where is the reducer?
4. Since we avoid using maven, we cannot count on it to generate the .jar, as we would normally do. You can instead build a jar manually using the File -> Export command, as in the following 2 figures. Remember to avoid inserting all the libraries in your jar, or you would end up producing a fat jar heavy to transfer. We need these libraries locally to compile, but they are already present in the classpath of the cluster and there is no need to include them in the jar again.


Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
EJB
Install
Java
JAR file
Javadoc
Runnable JAR file
Java EE
PHP
Plug-in Development
Remote Systems
< Back
Next >
Cancel
Finish



Specify the name of the output jar file (e.g., set the JAR file textbox to Exercise1-1.0.0.jar).

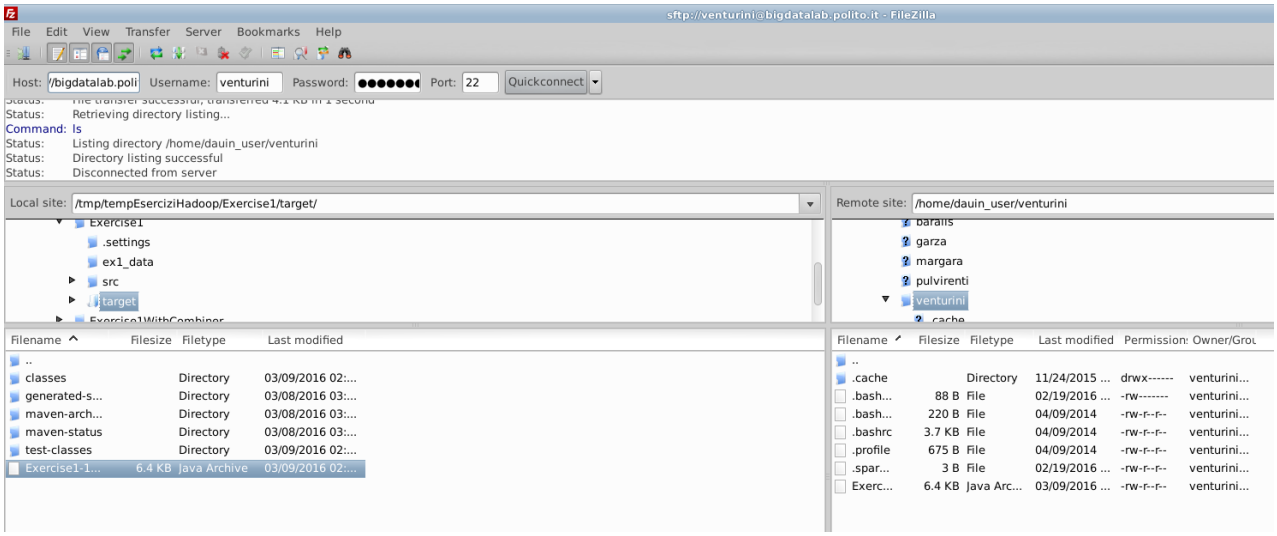
2. Transfer files with Filezilla

The objective of this task is to transfer some files from the local machine (i.e. the one you're working on right now) to a remote machine (e.g. the BigDataLab gateway). For this task, we are going to use Filezilla, but you can also use scp or rsync from the command line.

1. Connect to **bigdatalab.polito.it** [host] on port 22, using the username and password you have been given.

2. The Filezilla interface is divided in two sections:
    a. on the left side you have the file system of your local machine (LABINF PC),
    b. on the right side the file system of the remote machine (the bigdatalab.polito.it gateway, with its own remote disk).

    Please note that you don't see the HDFS distributed file system in Filezilla.



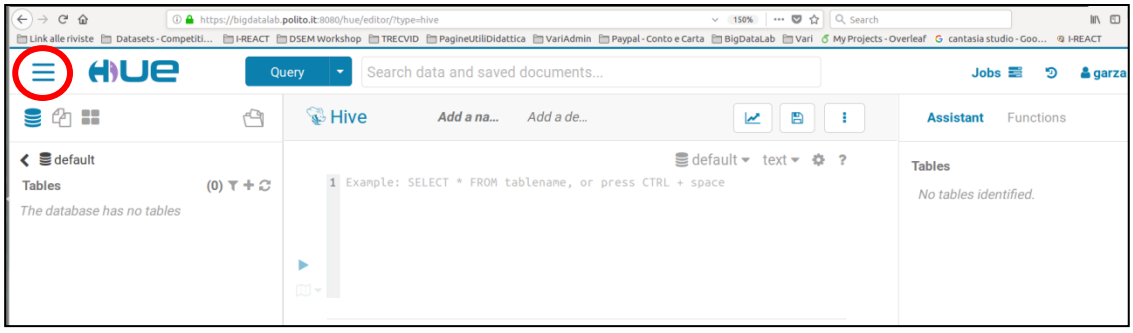
3. Locate on the local site your Exercise1-1.0.0.jar file (output of task 1), and a proper folder on the right (your home or a subfolder is fine). Right click on a file to find the upload command.

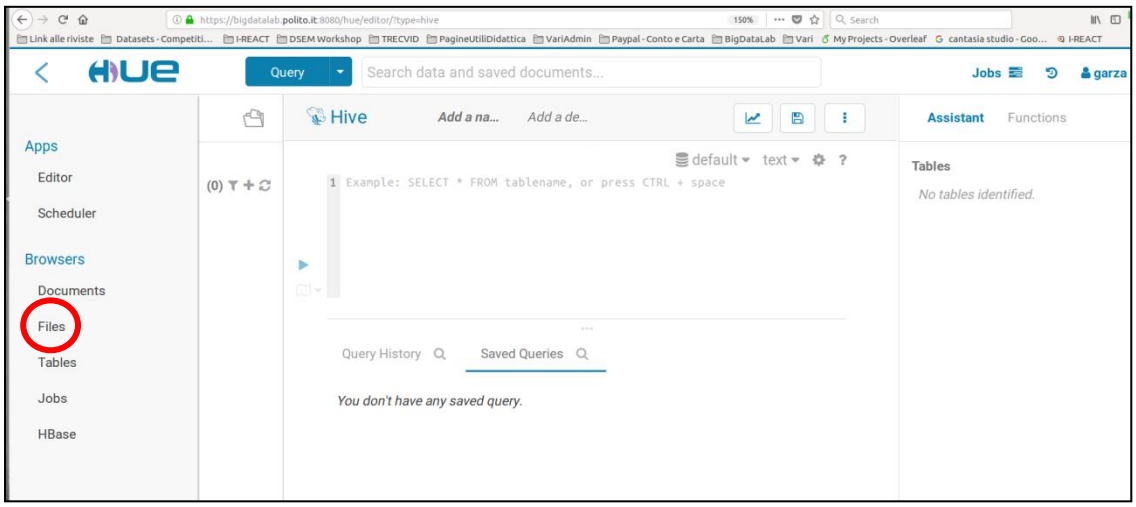
4. Check that the file was correctly transferred to the remote. You can either check the logs (above) or if the remote folder lists your file (on the right). In the figure, you see that a file, named Exercise1-1.0.0.jar, was copied to my home folder.

3. Manage HDFS through the HUE web interface

In this task, you will learn how to do basic management of the HDFS file system. To this goal, we will use a web interface called HUE. Again, you can do all the basic operations of this task on a command line; but this time, we advise you against this approach[1].

1. Go to https://bigdatalab.polito.it:8080 and login with your usual BigDataLab credentials.
2. Go to the "Browsers/Files" tab. You should find **your HDFS home**, as shown below. Note that this is not the same file system as in task 2 (i.e., the bigdatalab.polito.it physical disk), so you will find probably an empty folder now.[2]
   Your **HDFS home** is not located on the bigdatalab.polito.it machine, but is stored in the Hadoop cluster: the bigdatalab.polito.it is only a gateway that shows you the distributed HDFS contents.





3. Upload the sample file you have in ex1_data/document.txt, and check everything worked fine by opening it inside HUE. You should find two-three lines of sample text inside.

4. Find out on your own how to delete/move the files, or download them. It will be helpful in the next labs.

[1] You would need further configuration to access HDFS from a client on your local machine.
[2] If the difference between the two "homes" is not clear to you at this point, do not keep on. Spend some time to clearify your ideas.

4. Submit a job

Now we have everything we need to submit our sample application. It is finally time to open a command line. Using ssh, we will connect to the gateway we have used in task 2, where we should find the jar we copied at that time. From this gateway machine, we can (finally!) submit a job[3].

1. Connect, using SSH, on bigdatalab, with your usual BigDataLab credentials.

```
>> ssh sXXXXXX@bigdatalab.polito.it
```

2. Now that you have a terminal on the gateway, launch a job using this command:

```
>> hadoop jar Exercise1-1.0.0.jar
it.polito.bigdata.hadoop.exercise1.DriverBigData 1
your_input_file.txt ex1_out
```

   where:
   - "**Exercise1-1.0.0.jar**" is the JAR file on the remote bigdata.polito.it disk (you can see it in Filezilla or in the terminal via SSH)
   - "**your_input_file.txt**" is the input file on the cluster HDFS (you can see the HDFS file system through the HUE web interface), a relative path starts in your home in HDFS, you can also use an absolute path in HDFS
   - "**ex1_out**" is the output folder in HDFS, not on the bigdatalab.polito.it disk, you can see its content in HUE (a relative path starts in your home in HDFS)
3. Find your job on HUE interface (JobBrowser), check its status (submitted, running, failed or succeeded) and find its stderr and stdout. Find also all the running jobs at the present time (i.e. not only yours).
4. Find the output file on HDFS (from HUE file browser) and see the results for the wordcount.
5. Try to re-run the same job. Does it succeed this time? What's the problem?

[3] This gateway machine is simply the only machine in the cluster that can be accessed from outside, and the only one where users are allowed to launch jobs and manage HDFS.

5. Analyze the performances of your job

First, we need to setup some security configuration to access protected web interfaces on our cluster. First, we obtain a kerberos ticket that will grant us access to the system for a day; then, we will use Firefox to access the protected web interfaces (HUE is the only web UI you can access without completing this procedure).[4]
Keep in mind these operations for our next labs.

1. Open a new terminal on your local machine (do **not** use the previous terminal connected to bigdata.polito.it).
2. On the local terminal command line, type the command
   `kinit sXXXXXX`
   (then, you will be asked for your BigDataLab password)
3. Now you should have access to the full job history of YARN at
   https://ma1-bigdata.polito.it:19890/jobhistory/
4. How many mappers did your job instantiate? How many reducers?
5. Relaunch your job on a bigger file, that you can find on the cluster HDFS at the following path:
   /data/students/bigdata-01QYD/Lab1/finefoods_text.txt
   This a large collection of amazon reviews in the food category. Analyze the results. Can you understand any interesting facts from your results? Do you see any space for improvements in your analysis?
6. The following figure was done on a small sample of your data (10000 reviews). Is it consistent with what you found on the complete dataset? Do you think a small sample is enough to represent the whole?

[4] Details here: http://bigdata.polito.it/content/access-instructions

Bonus Task

A word count can be seen as a special case of an n-gram count, where n is equal to 1. n-grams, in our context, are sets of contiguous words of length n.

For example, in the sentence “She sells seashells by the seashore“, 2-grams are: She sells, sells seashells, seashells by, by the, the seashore.

Modify your word count program to count 2-grams frequencies. Consider each line of text as a separate document, as in the amazon reviews file: so, do not count as contiguous words on separate lines.

What is the time/space complexity of your program? How long would you expect it to run, compared to the simple word count? Try to run it on the toy text and on the amazon reviews.

Lab 2.

In this lab, you will write by your own a complete Hadoop application. Start by importing the template project available in Lab_Skeleton.zip. Once you have imported the template, modify the content of the classes to implement the application described in the following. The template contains the skeleton of a standard MapReduce application based on three classes: Driver, Mapper, and Reducer. Analyze the problem specification and decide if you really need all classes to solve the assigned problem.

From now on, keep in mind always (even if we do not explicitly ask for it) the complexity of your program. Specifically try to understand, before even submitting a job, what will be the effort you will require to the cluster in terms of time, network and I/0

- How many pairs and bytes will be read from HDFS?
- How many pairs and bytes will be emitted by the mappers and hence how many data will be sent on the network?

You can then check on HUE if you guessed correctly (more or less).

1. Filter an input dataset

If you completed Lab 1, you should now have (at least one) large files with the word frequencies in the amazon food reviews, in the format `word\tnumber`, where number is an int (a copy of the output of Lab 1 is available in the HDFS shared folder /data/students/bigdata-01QYD/Lab2/). You should also have realized that inspecting these results manually is not feasible. Your task is to write a Hadoop application to filter the content of the output of Lab 1 and analyze the filtered data.
The filter you should implement is the following:

- keep only the words that start with “ho”.

How large is the result of this filter? Do you need to filter more?

Modify the application in order to accept the beginning string as a command-line parameter. Execute the new version of the program to select the words starting with the prefix that you prefer.

Bonus task

If you completed the bonus task of lab 1, try your filter on the 2-grams you have generated.
If you did not complete the bonus task of lab 1, you can use the files available in the HDFS shared folder /data/students/bigdata-01QYD/Lab2BonusTrack/

What is the size of this new input dataset, compared to the simple word counts (1-grams) we used in the previous step? Did you really need the cluster to filter 1-grams? What about 2-grams?

Implement a new application that selects all the 2-grams that contain, at any position, the word “like” (i.e., “like” can be either the first or the second word of the selected 2-grams). What do you think will be, most likely, the other word?

Lab 3.

In this lab, we analyze a dataset containing the information about the items reviewed by some users of Amazon. Specifically, we will start looking at the connections between the reviewed items, and to do this we will use a dataset containing one line per reviewer. The first field of each line contains always the id of the reviewer (AXXXXXXXXX in the running example), followed by the list of all the products reviewed by her/him (BXXXXXXXXX in the running example).

Here is a sample of the first lines of such dataset:

```
A09539661HB8JHRFVRDC,B002R8UANK,B002R8J7YS,B002R8SLUY
A1008ULQSWI006,B0017OAQIY
A100EBHBG1GF5,B0013T5YO4
A1017Y0SGBINVS,B0009F3SAK
A101F8M8DPFOM9,B005HY2BRO,B000H7MFVI
A102H88HCCJJAB,B0007A8XV6
A102ME7M2YW2P5,B000FKGT8W
A102QP2OSXRVH,B001EQ5SGU,B000EH0RTS
A102TGNH1D915Z,B000RHXKC6,B0002DHNXC,B0002DHNXC,B000XJK7UG,B00008DFK5
A1051WAJL0HJWH,B000W5U5H6
A1052V04GOA7RV,B002GJ9JY6,B001E5E3JY,B008ZRKZSM,B002GJ9JWS
```

For the following exercise, you can use the sample dataset AmazonTransposedDataset_Sample.txt, which is available in the HDFS shared folder /data/students/bigdata-01QYD/Lab3. You can see the content of this file by using HUE (https://bigdatalab.polito.it:8080/hue).

Ex 1. “People also like…”

In this exercise, we try to build a very basic version of a recommending system. Your goal is to find the top 100 pairs of products most often reviewed (and so probably bought) together.

In this exercise, we consider two products as reviewed (i.e., bought) together if they appear in the same line of the input transposed file (the input file is /data/students/bigdata-01QYD/Lab3/AmazonTransposedDataset_Sample.txt). We ignore temporal constraints, so even if a decade or a thousand products have passed between the two reviews, we count the pair, as it represents anyway the tastes of a single user.

We suggest you to implement your application by using the template project contained in Lab3_Skeleton_Java8.zip. Import it in Eclipse by using the “Import” feature of Eclipse. The provided template already contains the skeleton of the Driver, Mapper, and Reducer classes. Fill out the missing parts and decide if you need one single job or two concatenated jobs.

Lab3_Skeleton.zip also contains two utility classes for your convenience. They should support you during the development of your solution.

- WordCountWritable
  - This class can be used to store a pair (word, count), where word is a String and count is an Integer.
  - The public WordCountWritable(String word, Integer count) constructor is used to create new WordCountWritable objects.
  - String getWord() and Integer getCount() are used to retrieve the value of word and count, respectively.
  - setWord(String value) and setCount(Integer value) are used to set the value of word and count, respectively.
  - This class implements the Comparable interface. Hence, the public int compareTo(WordCountWritable other) can be used to compare objects of this class.
  - This class implements the Writable interface. Hence, it can be used as data type of the value part of the pairs emitted by a mapper or a reducer if needed.

- TopKVector<T extends Comparable<T>>
  - This class can be used to store/manage in main-memory (in a local variable) the top-k objects of a set of objects. The type of managed objects is T, where T can be an arbitrary class implementing the Comparable interface (e.g., WordCountWritable).
  - The TopKVector(int k) method is the constructor used to create TopKVector objects. Each object of type TopKVector contains an internal Vector storing only the top-k objects among the set of objects of type T that are inserted in it. Initially, this internal vector is empty (i.e., initially the top-k vector contains no objects).
  - public void updateWithNewElement(T newElement) is used to insert a new object in the internal top-k vector of the TopKVector object on which the method is invoked. This method inserts the newElement object in the internal top-k vector if and only if it is in the top-k objects. Otherwise, it is discarded.
  - public Vector<T> getLocalTopK() returns a Vector<T> containing the top-k objects associated with the TopKVector object on which this method is invoked (i.e., it returns a copy of the internal vector containing only the top-k objects among the ones inserted by using the updateWithNewElement method).

The following snippet of code shows how to use these two classes. Decide in which parts of your solution you need these two classes.

```
// An example showing how to create an object that is used to store/manage a top-3 vector
// containing objects of type WordCountWritable.
// top3 is a local variable stored in the main-memory of the application
TopKVector<WordCountWritable> top3 = new TopKVector<WordCountWritable>(3);

// An example showing how to insert 5 objects of type WordCountWritable in the top3 local variable defined
// in the previous line of code.
// top3 automatically stores in its interval vector only the top-3 objects (based on the value of the count value) and // discards the objects that are not in the top-3 set.
top3.updateWithNewElement(new WordCountWritable(new String("p1,p2"), new Integer(4)));
```

```
top3.updateWithNewElement(new WordCountWritable(new String("p1,p3"), new
Integer(40)));
top3.updateWithNewElement(new WordCountWritable(new String("p2,p4"), new Integer(3)));
top3.updateWithNewElement(new WordCountWritable(new String("p5,p6"), new Integer(6)));
top3.updateWithNewElement(new WordCountWritable(new String("p15,p16"), new
Integer(1)));


// Retrieve the top-k objects from the local top3 variable
Vector<WordCountWritable> top3Objects = top3.getLocalTopK();

// Print the content of the top-3 selected objects on the standard output
for (WordCountWritable value : top3Objects) {
        System.out.println(value.getWord() + " " + value.getCount());
}

// The following is the output of this snippet of code if it is executed in a standalone Java
application
p1,p3 40
p5,p6 6
p1,p2 4
```

Lab 4

In this lab, we keep on our thoughts on the Amazon food dataset we have been using so far, and we start using the rating the users give to the products.

A rating is a number of stars between 1 and 5, where 5 is “I love it” and 1 is “I hate it”. There are a number of philosophical investigations and research papers on what the other rating really means. For some really critical users, 3 stars could be really a good rating, maybe the maximum they would ever give to a product; others instead, have never gone below 4, and only in exceptional cases, when the product bought was really unsatisfactory.

What we will try in this laboratory is to normalize the ratings according to the user’s proclivity to give a 5 star.

Let’s see an example:

	A1	A2	A3	A4	A5
B1		5		4	1
B2	3				
B3		5	4	5	4
B4				5	
B5		5		2	3

The columns of this matrix are the users, while the rows are the products. User A2 has given 3 reviews: to product B1, to B3 and to B5, all of which are 5 stars. A5 instead has given 1 star to B1, 4 stars to B3 and 3 stars to B5.

We can normalize this matrix subtracting, from each column, its mean value, obtaining thus:

	A1	A2	A3	A4	A5
B1		0		0	-1.67
B2	0				
B3		0	0	1	1.33
B4				1	
B5		0		-2	0.33

Now we see that, for example, A4 has given 1 more than its personal average to product B3, that is to say she likes it, while she has given 2 less than the average to B5, so she definitely does not like it. A5 instead was not so unsatisfied by B5, since she gave 0.33 more than its average rating to this product.

Now for each of the products we can compute the average of such normalized ratings, obtaining a "normalized average rating" for each product :

B1	-0.56
B2	0
B3	0.58
B4	1
B5	-0.56

Notice how the ranking of B5 was affected by this transformation.

Your task for this lab is to write a Hadoop Application to compute the normalized ratings of the products of the Amazon food dataset by considering the (sparse) products/users matrix based on the ratings available in the following HDFS file:
/data/students/bigdata-01QYD/Lab3/Reviews.csv

For the initial test of your application you can use the small sample dataset ReviewsSample.csv, which contains a set of reviews related to the same users, products, and ratings of the small example products/users matrix reported in this document.
ReviewSample.csv is available at the following address:
http://dbdmg.polito.it/wordpress/wp-content/uploads/2016/04/ReviewsSample.csv

Upload it in the HDFS file system before running your application.

Hints

Beware that the products/users matrix is sparse, i.e. many of its values are null/unknown. Hence, you can assume that the number of rating per user is a small. In your application try to exploit this fact, avoiding shuffling unnecessary (key, value) pairs on the network.