

# MapReduce Laboratory

In this laboratory students will learn how to use the Hadoop client API by working on a series of exercises:

- The classic Word Count and variations on the theme
- Design Pattern: Pair and Stripes
- Design Pattern: Order Inversion
- Join implementations

Note that the design patterns outlined above have been originally discussed in:

- Jimmy Lin, Chris Dyer, “Data-Intensive Text Processing with MapReduce”.

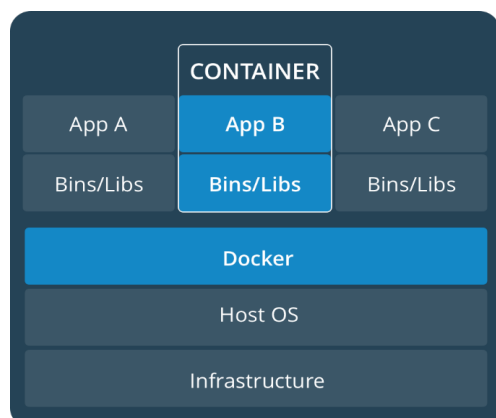
## 1 Hadoop on a container

### 1.1 What is a container?

Containers are a way to package software in a format that can run isolated on a shared operating system. Unlike virtual machines (VMs), containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

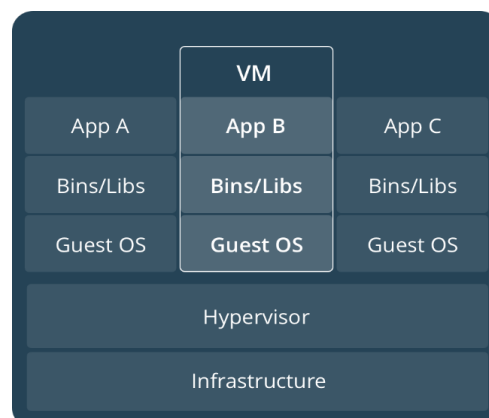
#### 1.1.1 Container vs Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware, containers are more portable and efficient.



### CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.



### VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

### 1.1.3 Docker

Docker is the leading Containers as a Service (CaaS) platform. Docker is an open source software containerization platform designed to make it easier to create, deploy and run various applications by using containers. According to the official Web page of Docker, its containers wrap up a piece of software into a complete file system that contains everything it needs to run: runtime, code, system tools or system libraries – anything we can install on a server. This ensures that it will always run the same, irrespective of the environment it is running in. Docker provides an additional layer of abstraction and even automation of virtualization at the operating system level on Linux. It actually uses different resource isolation features of the Linux kernel, like cgroups and kernel namespaces, and also a union-capable file system such as OverlayFS to allow independent ‘containers’ to run within a single Linux instance. This helps in avoiding the overhead of starting and then maintaining virtual machines, which significantly boosts its performance and also reduces the size of the application. The support of the Linux kernel for namespaces isolates an application’s view of the operating environment—including process trees, user IDs, network and mounted file systems —while the kernel’s cgroups helps in limiting resources, including the memory, CPU, block I/O and network.

### 1.1.4 Cloudera

Cloudera provides a scalable, flexible, integrated platform that makes it easy to manage rapidly increasing volumes and varieties of data in an enterprise. Cloudera products and solutions enable the user to deploy and manage Apache Hadoop and related projects, manipulate and analyze data, and keep that data secure and protected.

CDH is the Cloudera distribution of Apache Hadoop and other related open-source projects, including Cloudera Impala and Cloudera Search. CDH also provides security and integration with numerous hardware and software solutions.

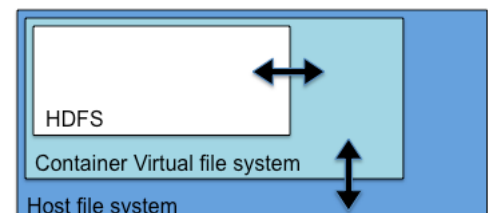
CDH is a complete, tested, and popular distribution of Apache Hadoop and related projects. CDH delivers the core elements of Hadoop – scalable storage and distributed computing – along with a Web-based user interface and vital enterprise capabilities. CDH is Apache-licensed open source and is a Hadoop solution that offers unified batch processing, interactive SQL and interactive search, and role-based access controls.

The Cloudera Docker image is a single-node deployment of the Cloudera open-source distribution, including CDH and Cloudera Manager. Users can utilize this environment to learn Hadoop, try new ideas, and test and demonstrate applications.

### 1.1.5 Storage management

As you can see in the picture, if you want to move a file from your local storage to the Hadoop Distributed File System (HDFS), you need to move the file from the *local storage* to the *container*, and then copy it to the *HDFS* partition. Clearly, the reverse path is also possible.

In addition, inside the Cloudera container runs a *web server* that provides an interface to HDFS: in this case, the user can access to the server from a browser on the host, and download / upload files to HDFS through the web server.



## 1.2 Download, install and use Docker

The following instructions are for Linux-based OS. For other Operating Systems (Windows, Mac OS) refer to the instructions on Docker’s website.

Install Docker with the following command:

```
sudo curl -sSL https://get.docker.com/ | sh
```

Note that the Docker version should be greater than 17. Use `docker --version` to check it!

To use Docker as a non-root user, you should now consider adding your user to the "docker" group with something like:

```
sudo usermod -aG docker [username]
```

### 1.2.1 Check that the daemon is running

```
service docker start
```

## 1.2.2 Importing the Cloudera QuickStart Image

You can import the Docker image by pulling it from the Docker Hub:

```
docker pull cloudera/quickstart:latest
```

## 1.2.3 First run

```
docker run --hostname=quickstart.cloudera --privileged=true -t -i -p 8888:8888 -p 80:80  
-p 7180:7180 --name cloudera cloudera/quickstart /usr/bin/docker-quickstart
```

<code>--hostname=quickstart.cloudera</code>	Required: Pseudo-distributed configuration assumes this hostname.
<code>--privileged=true</code>	Required: For HBase, MySQL-backed Hive metastore, Hue, Oozie, Sentry, and Cloudera Manager.
<code>-t</code>	Required: Allocate a pseudoterminal. Once services are started, a Bash shell takes over. This switch starts a terminal emulator to run the services.
<code>-i</code>	Required: If you want to use the terminal, either immediately or connect to the terminal later.
<code>-p 8888</code>	Recommended: Map the Hue port in the guest to another port on the host.
<code>-p [PORT]</code>	Optional: Map any other ports (for example, 7180 for Cloudera Manager, 80 for a guided tutorial).
<code>-d</code>	Optional: Run the container in the background.

Once the first run is completed, the shell will be the container shell. Just exit, and the container will **stop**.

## 1.2.4 Running the container (after the first installation)

- List all the containers:

```
docker ps -a
```

- Run the desired container by ID:

```
docker start [CONTAINER ID]
```

At this point you will have the container running. In order to interact with the container, you can start a bash:

```
docker exec -it cloudera /bin/bash
```

## 1.2.5 Stop the container

- List all the containers:

```
docker ps -a
```

- Stop the desired container by ID:

```
docker stop [CONTAINER ID]
```

## 1.2.6 Stop Docker

To stop the docker service, just write the following command:

```
service docker stop
```

## 1.3 Copy data to and from the container

To copy your file(s) between the local filesystem and the container, you need to use the following command from the host shell:

- Copy from container to local filesystem

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH
```

- Copy from local filesystem to container

```
docker cp [OPTIONS] SRC_PATH CONTAINER:DEST_PATH
```

Options:

<code>--archive , -a</code>	Archive mode (copy all uid/gid information)
<code>--follow-link , -L</code>	Always follow symbol link in SRC_PATH

The “CONTAINER:” is the name of the container (e.g., “cloudera:”).

## 1.4 Manage the Cloudera container from a browser

The Cloudera container runs a web server (called *Hue*) that provides information about the container and about CDH (i.e., Cloudera distribution of Hadoop). Use the following links (user: cloudera, password: cloudera):

<http://localhost/#/>  
<http://localhost:8888>

*Hue* lets you browse HDFS, Jobs, run Hive, Pig and Cloudera Impala queries, manage the Hive Metastore, HBase, Sqoop, ZooKeeper, MapReduce jobs, and create and schedule workflows with Oozie.

## 2 Alternative: Virtual Machine

### 2.1 VM details

- It is preconfigured with all the software needed to get through the assignments.
- It has an Ubuntu 12.04 LTE (64bit) with XFCE Desktop Environment
- It requires  $\geq$  2GB of memory
- It has a single account:
  - username: student
  - password: password
  - Notice that login is password less and root commands can be run using sudo which is password less as well.

**IMPORTANT: Shut down the VM once finished the session (do not close abruptly the window)**

### 2.2 Run the VM on the laptop

Download the image tarball from

<http://tinyurl.com/mr-lab-bigfoot>

and uncompress it

The image it can be used either VMplayer or Virtualbox

VMplayer:

- Windows/Linux: download the freeware player from here
- Mac OS: unfortunately there is no freeware player, but it can be used VMware Fusion which has a 30-day trial.

Virtualbox:

- The image can be loaded in Virtualbox with no issues. Be sure to select the file HadoopVM.vmdk as a virtual hard-drive (and not one of the HadoopVM-sXXX.vmdk files).
- To install guest additions, once the image is running:
  - `sudo apt-get install dkms build-essential`
  - download the additions (devices → install guest additions)
  - `sudo su -`
  - `cd /media/VIRTUALBOXADDITION_version`
  - `./VBoxLinuxAddition.run`

## 2.3 Copy data to and from the VM

File can be sent or retrieved from the VM with a shared folder between the VM and the host machine.

## 2.4 Web interfaces: Monitor job progress

Hadoop publishes some web interfaces that display JobTracker and HDFS statuses. Inside the VM, you can run a browser and access the following links:

- Jobtracker Web Interface: <http://127.0.0.1:50030/>
- NameNode Web Interface: <http://127.0.0.1:50070/>

Note that the above addresses works when you can contact the machine where Hadoop runs (in this case, they are visible inside the VM, not outside). In case of a real cluster, one should use the IP address of the Hadoop gateway.

## 3 Utilities

In order to run a job, the process to follow is:

- Create the .java file (with a text editor or Eclipse);
- Compile the file and create the jar;
- Transfer the file to the container (alternatively, on the VM); in real life, if there is a cluster, the jar will be transferred to the cluster gateway node;
- The HDFS on the container (alternatively, on the VM, or on the real cluster) will have the data to analyze;
- From the container (alternatively, from a shell of the VM, or from a shell on the gateway machine of the cluster) launch the job using the jar uploaded.

Another option is to transfer the source file on the container (or VM, or cluster gateway), and compile the source code (and create the jar) from the container (or VM, or cluster gateway), and then launch the job.

For correctly compiling the code on the host, the following set of Hadoop-specific libraries are needed:

- `/usr/lib/hadoop/hadoop-common.jar`
- `/usr/lib/hadoop/hadoop-annotations.jar`
- `/usr/lib/hadoop-0.20-mapreduce/hadoop-core-2.6.0-mr1-cdh5.7.0.jar`
- `/usr/lib/hadoop-hdfs/hadoop-hdfs.jar`
- `/usr/lib/hadoop-mapreduce/log4j-1.2.17.jar`

These libraries can be found on the Internet, or they can be transferred from the container. First create a “mylib” directory somewhere on you machine, then use the following command

```
docker cp --archive -L cloudera:/usr/lib/hadoop/hadoop-common.jar ./mylib/
```

Repeat the process for the other jars.

NOTE: The container version of Hadoop supports Java 1.7. If you have installed Java 1.8, you need to compile the code by specifying the use of Java 1.7. You will see a warning, but the code should run without problems. If you compile the source code on the container, then you do not need to specify the java version, since the container has Java 1.7.

## 3.1 Writing code with a text editor

For the sake of example, we will use the “word count” source code provided on the course website. We assume that you have written the source code with any text editor, and now you have to compile it via command-line.

- Create the build directory

```
mkdir -p build
```

- Compile (note that we have forced the use of Java 1.7 through the options “-source” and “-target”)

```
javac -cp "mylib/*" -source 1.7 -target 1.7 WordCount.java -d build -Xlint
```

- Create a JAR file for the WordCount application.

```
jar -cvf wordcount.jar -C build/ .
```

- Move the jar to the container

```
docker cp --archive -L ./wordcount.jar cloudera:/<PATH>/wordcount.jar
```

## 3.2 Writing code with Eclipse

Eclipse is an integrated development environment (IDE) that may help managing the code.

To this aim, when you create a project, you have add the Hadoop-specific jars. To create a new Java Project in Eclipse:

- Inside Eclipse select the menu item **File > New > Project ...** to open the **New Project** wizard.
- Select **Java Project** then click **Next** to start the **New Java Project**
- Type a name for your new project, such as "mr-lab"
- Ensure to use *JavaSE-1.7* as JRE, then click on **Next**
- Go on **Libraries** tab, and click on **Add External Jars**
- Add the jars previously downloaded on your “mylib” directory:
  - ~/mylib/hadoop/hadoop-common.jar
  - ~/mylib /hadoop/hadoop-annotations.jar
  - ~/mylib /hadoop-0.20-mapreduce/ hadoop-core-2.6.0-mr1-cdh5.7.0.jar
  - ~/mylib /hadoop-hdfs/hadoop-hdfs.jar
  - ~/mylib /hadoop-mapreduce/log4j-1.2.17.jar
  - Click on **Finish**

Now you have a new project, from which you can create the jar that will be run by Hadoop. In order to create the jar in Eclipse, you should:

- Select the menu item **File > Export**
- Type **JAR**, select **JAR file** and click on **Next**
- In the textbox **Select the export destination**, type the name of the new **JAR** you want to export, i.e. 'wordcount.jar'
- Type **Finish**.

Now you can move the jar to the container

```
docker cp --archive -L <ECLIPSE_WORKSPACE>/wordcount.jar cloudera:/<PATH>/wordcount.jar
```

### 3.3 Compile the code within the container

As a last option, you can copy the source file `WordCount.java` on the container and compile it there. In this case, you do not need to download the libraries on the host, since such libraries are already present in the container.

- Copy `WordCount.java` from your local storage to the container

```
docker cp --archive -L ./WordCount.java cloudera:/<PATH>/WordCount.java
```

- Compile the `WordCount` class

```
mkdir -p build
```

```
javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* WordCount.java -d build -Xlint
```

- Create a JAR file for the `WordCount` application.

```
jar -cvf wordcount.jar -C build/ .
```

### 3.4 How to launch a job

Once you have created the jar file (from the command line or from Eclipse), and transferred the jar to the container, the jar can be launched with Hadoop.

On the container bash, type:

```
hadoop jar <jarname.jar> <fully.qualified.class.Name> <Parameters>
```

For example, for running the *WordCount* exercise, you will type:

```
hadoop jar wordcount.jar org.myorg.WordCount /user/cloudera/wordcount/input  
/user/cloudera/wordcount/output
```

Note that you need to specify a *non existing* output directory, or to delete it before running the job. We will show in the first exercise the details of the `WordCount` file and the input/output directories used.

### 3.5 Navigate HDFS – Transfer data to and from HDFS

The HDFS client can talk to the namenode to obtain information on the distributed file system, and to make a number of operations. The basic command is

- `hadoop fs [option]`
- where option can be
  - `-ls <path>` → Displays the content of the directory `<path>`
  - `-put <localsrc> <dst>` → Copies files from the local file system to hdfs
  - `-get <src> <localdst>` → Copies files from hdfs to the local file system
  - `-cat <file>` → Displays `<file>` on the terminal (copies to stdout)
  - and many others... see the Hadoop FS shell commands official documentation.

Alternatively, one can use the web interface to interact with the namenode.

## 4 Exercises

Note, exercises are organized in ascending order of difficulty.

### 4.1 Exercise 1 – Word Count

Count the occurrences of each word in a text file. This is the simplest example of MapReduce job: in the following we illustrate three possible implementations.

- **Basic:** the map method emits 1 for each word. The reduce aggregates the ones it receives from mappers for each key it is responsible for and save on disk (HDFS) the result
- **In-Mapper Combiner:** instead of emitting 1 for each encountered word, the map method (partially) aggregates the ones, and emit the result for each word (at the end of the Map)
- **In-Memory Combiner:** use the method “Setup” to create a data structure that keeps track of the words seen by the mappers; then use the method “Cleanup” to emit all the words at the end of the block

#### Instructions – Basic

For the **basic** version, the student has to refer to the *WordCount.java* provided on the course website. The archive contains also the text file to analyze (quote.txt).

Since the source file has been given, for this exercise the student should look at the code and understand the meaning of each line.

In the following, we will see how to run this example (please, refer to the “Utilities” section for the meaning of the different steps).

- First, put the data on HDFS
  - Using a host shell, transfer the “quote.txt” file from local filesystem to the container

```
docker cp -a -L ./ quote.txt cloudera:/<PATH>/quote.txt
```
  - Using the container shell, create a directory in HDFS that will contain the file (note that we need more than one step to create subdirectories)

```
hadoop fs -mkdir /user/myname/  
hadoop fs -mkdir /user/myname/wordcount/  
hadoop fs -mkdir /user/myname/wordcount/input/
```
  - Using the container shell, transfer the “quote.txt” from the container to the HDFS directory

```
hadoop fs -put quote.txt /user/myname/wordcount/input/
```
- Compile the *WordCount.java*, create the jar file (wordcount.jar) and move the jar file to the container as explained in Sect. 3.1 (command line) or 3.2 (Eclipse)
  - Note that *WordCount.java* belongs to a package (org.myorg): you need to refer to such a package when launching the job
- Run the Hadoop job
  - Note that we use, as input directory, the HDFS directory where we have put the “quote.txt” file, and we specify a non-existing directory for the output

```
hadoop jar wordcount.jar org.myorg.WordCount /user/myname/wordcount/input  
/user/myname/wordcount/output
```
- Check the output
  - You can read the files in the HDFS output directory with the browser (using Hue), or, using the container shell, you can stream the file to stdout (of the container) with “`hadoop fs -cut <file_to_read>`”, or you can transfer the files from HDFS to the container local files system.
  - The output should look like:

```
      2  
Cook   1  
Programmers      1  
Rich    1  
So       1  
Universe 3  
a        1
```



```

and      2
are      1
better   2
bigger   2
create   2
far      1
idiot    1
idiots.   1
in       1
is       2
programs,    1
proof    1
race     1
the      3
to       2
trying   1
winning. 1
with     1
while    1

```

Note the output messages provided by Hadoop, with interesting information about the job. Copy and paste these information on a file, since you will compare them with the corresponding output in the next exercises.

Exercises:

- Modify the file such that it is possible to specify the number of reducers when the job is launched, and observe the output;
- Add another file in the input directory and observe (and compare) the job counters with the basic case.

## Instructions – In-mapper and In-memory

Starting from the basic example, the student should create new versions with the *in-mapper* and *in-memory* design pattern. For the latter, the student will need to use the “setup” and cleanup” methods.

Once completed, the student should export the jar files and execute them on the container. The student should compare the output messages provided by Hadoop with the different versions of the WordCount (basic, in-mapper and in-memory).

## 4.2 Exercise 2 – Term co-occurrences

In the following exercise, we need to build the term co-occurrence matrix for a text collection. A co-occurrence matrix is a  $n \times n$  matrix, where  $n$  is the number of unique words in the text. For each couple of words, we count the number of times they co-occurred in the text in the same line.

### 4.2.1 Pairs Design Pattern

The basic (and maybe most intuitive) implementation of this exercise is the *Pair*. The basic idea is to emit, for each couple of words in the same line, the couple itself (or *pair*) and the value 1. For example, in the line `w1 w2 w3 w1`, we emit `(w1,w2):1`, `(w1,w3):1`, `(w2,w1):1`, `(w2,w3):1`, `(w2,w1):1`, `(w3,w1):1`, `(w3,w2):1`, `(w3,w1):1`.

In this exercise, we need to use a composite key to emit an occurrence of a pair of words. The student will understand how to create a custom Hadoop data type to be used as key type.

A Pair is a tuple composed by two elements that can be used to ship two objects within a parent object. For this exercise the student has to implement a TextPair, that is a Pair that contains two words.

## Instructions

There are two files related to this exercise:

- *TextPair.java*: data structure to be implemented by the student. Besides the implementation of the data structure itself, the student has to implement the serialization Hadoop API (write and read Fields).
- *Pair.java*: the implementation of a pair example using *TextPair.java* as datatype.

Once completed and compiled, create the jar and run it on the container using the text analyzed in the word count exercise. As done before, save the output information, since they will be compared with the following exercise.

## 4.2.2 Stripes Design Pattern

This approach is similar to the previous one: for each line, co-occurring pairs are generated. However, now, instead of emitting every pair as soon as it is generated, intermediate results are stored in an associative array. We use an associative array, and, for each word, we emit the word itself as key and a *Stripe*, that is the map of co-occurring words with the number of associated occurrence.

For example, in the line `w1 w2 w3 w1`, we emit:

```
w1:{w2:1, w3:1}, w2:{w1:2,w3:1}, w3:{w1:2, w2:1}, w1:{w2:1, w3:1}
```

Note that, instead, we could emit also:

```
w1:{w2:2, w3:2}, w2:{w1:2,w3:1}, w3:{w1:2, w2:1}
```

In this exercise the student will understand how to create a custom Hadoop data type to be used as value type.

### Instructions

There are two files related to this exercise:

- *StringToIntAssociativeArrayWritable.java*: the data structure file, to be implemented
- *Stripes.java*: the MapReduce job, that the student must implement using the *StringToIntMapWritable* data structure

Once completed and compiled, create the jar and run it on the container using the text analyzed in the word count exercise. Compare the output messages with messages provided when running the Pair implementation.

## 4.3 Exercise 3 – Relative term co-occurrence and Order Inversion Design Pattern

In this example we need to compute the co-occurrence matrix, like the one in the previous exercise, but using the relative frequencies of each pair, instead of the absolute value. Practically, we need to count the number of times each pair  $(w_i, w_j)$  occurs divided by the number of total pairs with  $w_i$  (marginal).

The student has to implement the `Map` and `Reduce` methods and the special partitioner (see `OrderInversion#PartitionerTextPair` class), which apply the partitioner only according to the first element in the Pair, sending all data regarding the same word to the same reducer. Note that inside the `OrderInversion` class there is a field called `ASTERISK` which should be used to output the total number of occurrences of a word.

### Instructions

There is one file for this exercise called `OrderInversion.java`. The `run` method of the job is already implemented, the student should complete the mapper, the reducer and the partitioner, as explained in the TODOs. Once completed and compiled, create the jar and run it on the container using the text analyzed in the word count exercise. Compare the output messages with messages provided with the previous approaches.

## 4.4 Exercise 4 – Joins

In MapReduce the term join refers to merging two different dataset stored as unstructured files in HDFS. As for databases, in MapReduce there are many different kind of joins, each with its use-cases and constraints. In this laboratory the student will implement the **Reduce-Side Join**: the map phase tags each record such that records of different inputs that have to be joined will have the same tag. Each reducer will receive a tag with a list of records and perform the join.

### Instructions

The student needs to find the two-hops friends, i.e. the friends of friends of each user, in the twitter dataset. In particular, the student needs to implement a self-join, that is a join between two instances of the same dataset, on the twitter graph. To test the code use the file `twitter-small.txt`, provided (to be transferred to HDFS). To run the final version of the job, the student can use a bigger file, `twitter-big-sample.txt`. Both files contain lines in the form `userid friendid`.

The student should use the file *ReduceSideJoin.java* as starting point.

## 4.5 Exercise 5 – PageRank

In this example, we compute the simple version of the PageRank, i.e., we do not consider the jump factor and the presence of sink nodes (every node has at least one output link, and the graph is completely connected). The graph is described through its adjacency list: in particular, the starting input file has the following format for each line:

```
nodeID    currentPageRank    neighNodeID1    neighNodeID2    neighNodeID3    ...
```

where “nodeID” is the unique identifier of the node, “currentPageRank” is the current value of the PageRank (initially set to the inverse of the number of nodes), and then there are the identifiers of the output links (the graph is directed, i.e., the adjacency matrix is not symmetric). The files of two different graphs can be downloaded from the course webpage (and should be stored in HDFS).

The student has to implement the **Map** and **Reduce** methods for the simplified PageRank computation. The `run` method of the job should launch a finite number of iterations (given as input parameter). At each iteration, the intermediate directory should be removed, so that, at the end, there will be only the initial directory and the final one. Note that, since the output of a MapReduce computation is a directory, and the PageRank computation is iterative, then the input (even at the beginning) should be always a directory. The student, therefore, should put the files describing a graph in separated directories.