LAB: Working with Hadoop and MapReduce

Outline

HDFS

- creating folders
- copying files
- ...

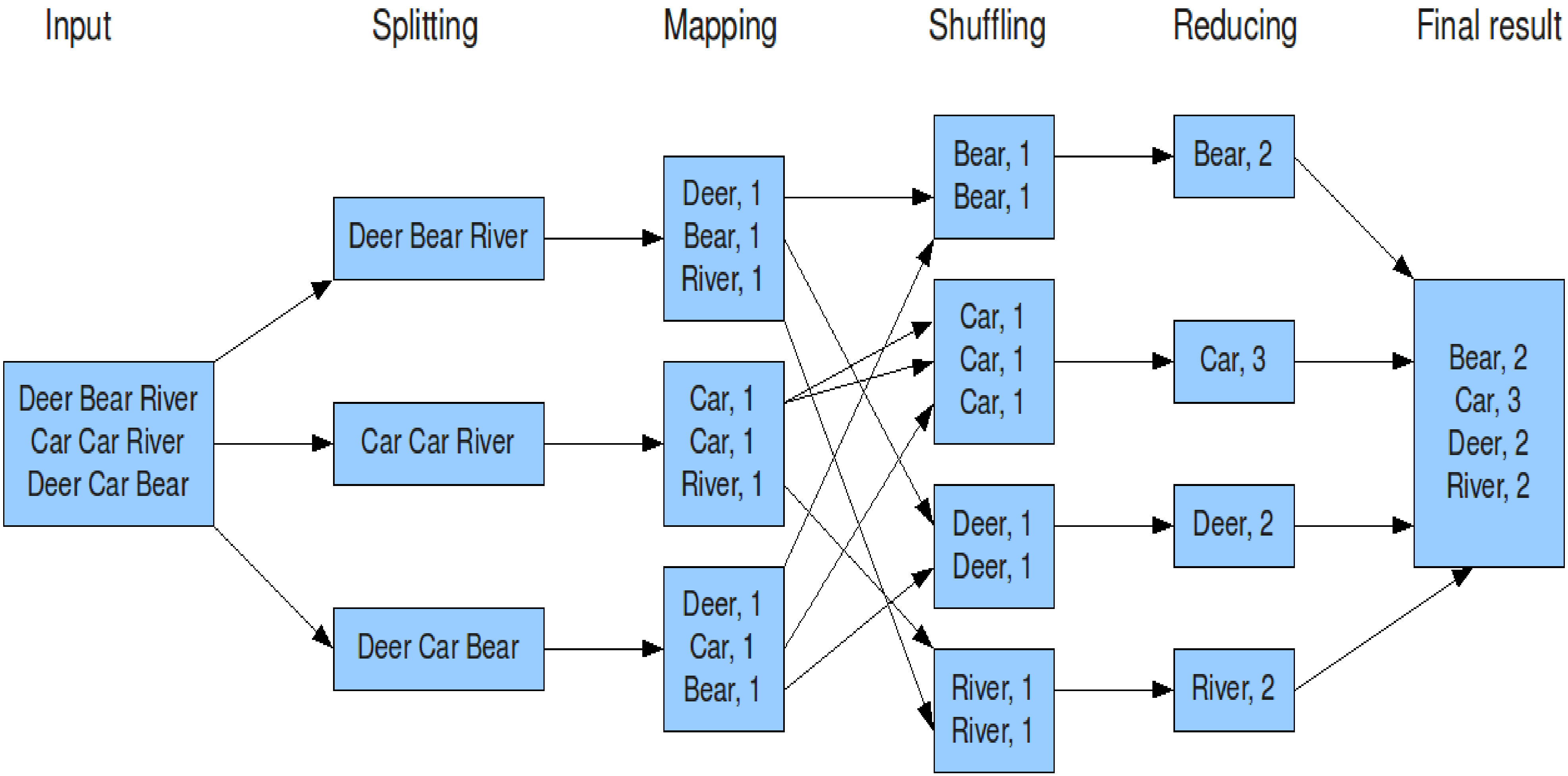
Hadoop Programming with Java

- WordCount
- MaxTemp

Hadoop Programming with Python

- WordCount
- MaxTemp
- WordLength
- InvertedIndex
- LinkCount

Reminder



Input

Deer Bear River
Car Car River
Deer Car Bear

Splitting

Deer Bear River

Car Car River

Deer Car Bear

Mapping

Deer, 1
Bear, 1
River, 1

Car, 1
Car, 1
River, 1

Deer, 1
Car, 1
Bear, 1

Shuffling

Bear, 1
Bear, 1

Car, 1
Car, 1
Car, 1

Deer, 1
Deer, 1

River, 1
River, 1

Reducing

Bear, 2

Car, 3

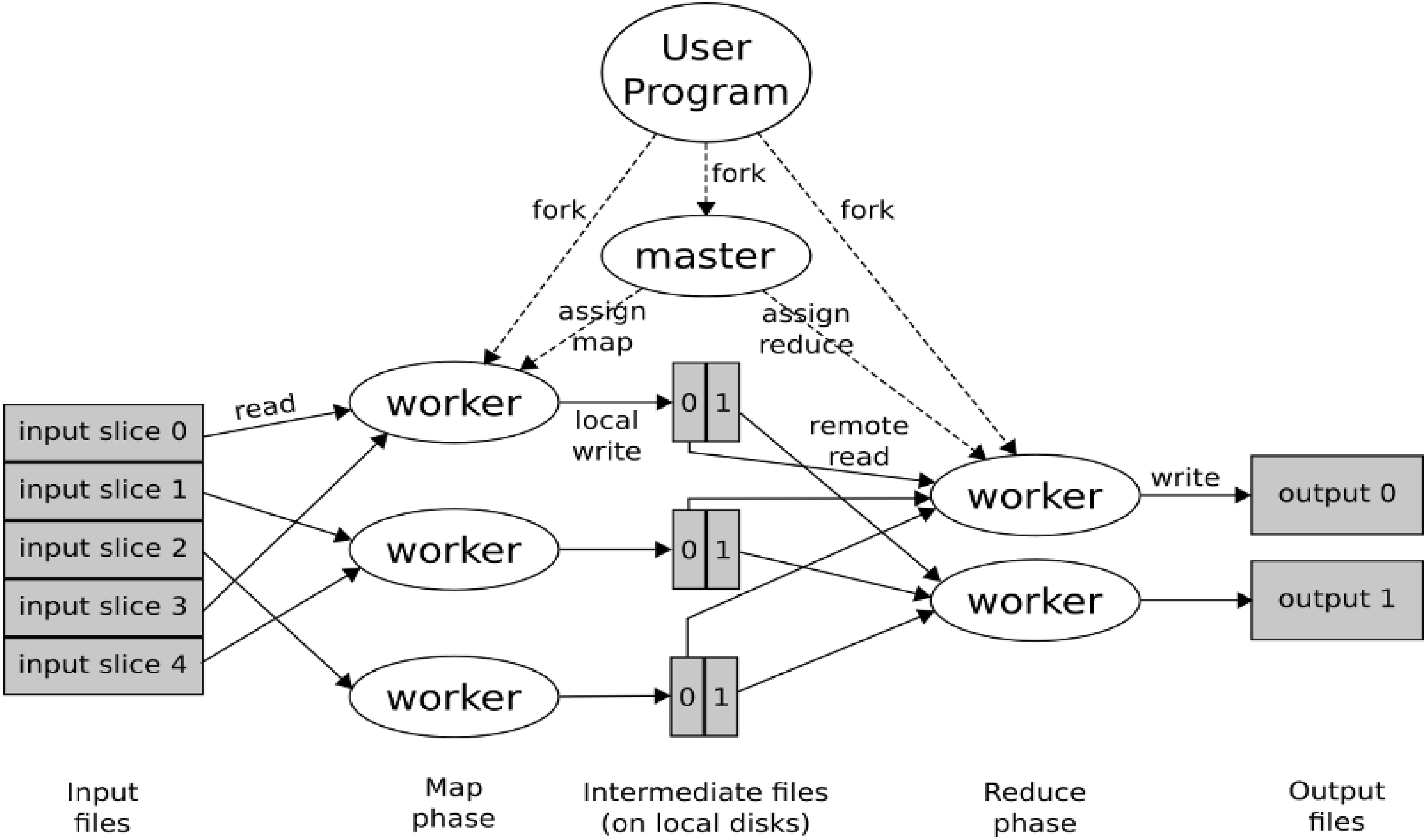
Deer, 2

River, 2

Final result

Bear, 2
Car, 3
Deer, 2
River, 2

Reminder



HDFS

HDFS

Target

To be able to write distributed programs over a **Hadoop cluster**.

The examples are simple for illustration purposes BUT the process we will follow is the same either we have an easy or a difficult problem.

Infrastructure

The cluster at LiX is composed of 32 physical nodes, of which one is the **MASTER** and the others are **SLAVES**.

All machines are running **CentOS Linux** and they have 16GB or RAM and 2TB of disk space.

The cluster is used for **training purposes**.

Prerequisites

To have an Internet connection.

To be able to login to

`master-bigdata.polytechnique.fr`

To have at least a **small experience** with programming.

Prerequisites

Login to

`master-bigdata.polytechnique.fr`

using your username/password

Copy the file **`/opt/hadooplab.tar.gz`** in your home directory by executing:

```
cp /opt/hadooplab.tar.gz ~
```

Extract the archive:

```
tar xvf hadooplab.tar.gz
```

HDFS

To get a list of all available commands

```
hadoop fs -help
```

HDFS

Listing files

```
hadoop fs -ls /
hadoop fs -ls /user
hadoop fs -ls /user/username
```

Useful: If you do not specify a path, HDFS will execute the command in your HDFS home directory which is /user/username

HDFS

Show file contents

```
hadoop fs -cat /dssp/data/leonardo/leonardo.txt
```

HDFS

File copy

```
hadoop fs -cp /dssp/data/leonardo/leonardo.txt /user/username/leonardo.txt
```

OR

```
hadoop fs -cp /dssp/data/leonardo/leonardo.txt leonardo.txt
```

View the file

```
hadoop fs -cat /user/username/leonardo.txt
```

OR

```
hadoop fs -cat leonardo.txt
```

Delete the file

```
hadoop fs -rm /user/username/leonardo.txt
```

OR

```
hadoop fs -rm leonardo.txt
```

HDFS

Creating and deleting directories

```
hadoop fs -mkdir /user/username/d1
```

OR

```
hadoop fs -mkdir d1
```

```
hadoop fs -rmdir /user/username/d1
```

OR

```
hadoop fs -rmdir d1
```

HDFS

Delete a directory and ALL CONTENTS

```
hadoop fs -rm -r /some-directory
```

BE VERY CAREFUL WHEN YOU USE IT!

HDFS

Putting/getting files to/from HDFS

```
hadoop fs -put fname.txt  /user/username/input
```

OR

```
hadoop fs -put fname.txt input
```

```
hadoop fs -get /user/username/input/fname.txt
```

OR

```
hadoop fs -get fname.txt
```

HDFS Preparation

Input data

All necessary input data files we are going to use are **already located** in HDFS in the directory:

```
/dssp/data
```

List the data directories by executing

```
hadoop fs -ls /dssp/data
```

HDFS Preparation

We will create an output directory to store the output of hadoop jobs

```
hadoop fs -mkdir /user/username/output
```

OR

```
hadoop fs -mkdir output
```

Hadoop with Java

We will focus on two examples of Hadoop jobs using the Java programming language.

WordCount: given a collection of text documents, find the number of occurrences of each word in the collection.

MaxTemp: given a file containing temperature measurements, find the maximum temperature recording per year.

WordCount: the mapper

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map (Object key, Text value, Context context)
        throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set (itr.nextToken());
            context.write (word, one);
      }
    }
  }
```

WordCount: the reducer

```
public static class IntSumReducer extends
   Reducer<Text,IntWritable,Text,IntWritable> {

    private IntWritable result = new IntWritable();


    public void reduce(Text key, Iterable<IntWritable> values, Context context)

      throws IOException, InterruptedException {

      int sum = 0;

      for (IntWritable val : values) {

        sum += val.get();

      }

      result.set(sum);

      context.write(key, result);

    }

  }
```

WordCount: main function

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

WordCount: compiling the code

Go inside the java-wordcount folder, by executing the following command from your home folder:

```
cd ~/hadooplab/java-wordcount
```

The relevant code is contained in the file

`WordCount.java`

WordCount: compiling the code

To compile the code run the command:

```
javac -classpath "$(yarn classpath)" WordCount.java
```

The file **`WordCount.class`** must have been produced.

WordCount: building the jar

We will create the file

`wc.jar`

Please execute

```
jar cf wc.jar WordCount*.class
```

Everything is set! Lets run the job on the cluster.

WordCount: running the job

Execute the following command:

input output

```
hadoop jar wc.jar WordCount /dssp/data/leonardo /user/username/output/wcj
```

Put your **username** here

OR

```
hadoop jar wc.jar WordCount /dssp/data/leonardo output/wcj
```

WordCount: exploring the results

```
hadoop fs -ls output/wcj
```

You should see something like this

```
-rw-r--r--   3 a.papadopoulos a.papadopoulos          0 2015-05-14 18:32 /user/a.papadopoulos/out2/_SUCCESS
-rw-r--r--   3 a.papadopoulos a.papadopoulos   53163233 2015-05-14 18:32 /user/a.papadopoulos/out2/part-r-00000
```

WordCount: exploring the results

Examine the last lines of the output:

```
hadoop fs -tail output/wcj/part-r-00000
```

Hadoop with Python

In this part, we focus on the Python language. We will discuss several different problems:

WordCount: given a collection of text documents, find the number of occurrences of each word in the collection.

MaxTemp: given a file containing temperature measurements, find the maximum temperature recording per year.

WordLength: find the average word length in a collection of document beginning by each letter.

InvertedIndex: given a collection of text documents, determine the set of documents that contain each unique word of the collection.

LinkCount: given a graph, determine the number of outgoing links for each node.

Hadoop Streaming

This is a tool that Hadoop provides to support code writing in any language (we will use it for Python)

Assumptions: Mappers should read from **`stdin`** and write to **`stdout`**. Reducers should read from **`stdin`** and write to **`stdout`**.

WordCount: the mapper

```
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py

        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

WordCount: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
  streaming-3.1.4.jar -file ./mapper.py –mapper./mapper.py
  -file ./reducer.py -reducer ./reducer.py -input
  /dssp/data/leonardo -output output/wcp
```

WordCount: exploring the results

To view the last lines of the output use **-tail**

```
hadoop fs -tail output/wcp/part-00000
```

To view the whole output file use **-cat**

```
hadoop fs -cat output/wcp/part-00000
```

WordCount: set number of reducers

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
  streaming-3.1.4.jar

    -D mapred.reduce.tasks=4

    -file ./mapper.py -mapper ./mapper.py

    -file ./reducer.py -reducer ./reducer.py

    -input /dssp/data/leonardo

    -output output/wcp2
```

WordCount: set number of reducers

Check the number of output files

```
hadoop fs -ls output/wcp2
```

How many output files are there?

WordLength: the problem

Given a document collection find the average word length beginning with each upper case letter.

WordLength: the mapper

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        if (word[0]>='A') and (word[0] <= 'Z'):
            print '%s\t%s' % (word[0].upper(), len(word))
```

WordLength: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
  streaming-3.1.4.jar
    -file ./mapper.py -mapper ./mapper.py
    -file ./reducer.py -reducer ./reducer.py
    -input /dssp/data/leonardo
    -output output/wlp
```

WordLength: exploring the results

To view the whole output file use **-cat**

```
hadoop fs -tail output/wlp/part-00000
```

Inversion: the problem

Given a collection of text documents, for each word determine the list of document ids containing the word.

The resulting data structure is known as the **inverted index** and the process of creating it is called **inversion**.

Inversion: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
  streaming-3.1.4.jar

    -file ./mapper.py -mapper ./mapper.py

    -file ./reducer.py -reducer ./reducer.py

    -input /dssp/data/leonardo

    -output output/invp
```

Inversion: exploring the results

To view the whole output file use **-cat**

```
hadoop fs -tail output/invp/part-00000
```

LinkCount: the problem

Given a graph G(V,E) determine the number of links for each node of the graph.

Web analogy: links are URLs

Input format:

```
1 4
1 5
2 6
6 8
```

...

LinkCount: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
  streaming-3.1.4.jar
    -file ./mapper.py -mapper ./mapper.py
    -file ./reducer.py -reducer ./reducer.py
    -input /dssp/data/enron
    -output output/lcp
```

LinkCount: exploring the results

To view the whole output file use **-cat**

```
hadoop fs -tail output/lcp/part-00000
```

Testing the Code Locally

Since hadoop streaming requires to read from stdin and write to stdout, **we can test our code without submitting the job to the cluster**.

Go into the python-wordcount directory and run the following command:

```
cat ./leonardo.txt | python
 mapper.py | sort | python
 reducer.py > output.txt
```