

LAB: Working with HDFS and MapReduce

DSSP



Outline

HDFS

- creating folders
- copying files
- ...

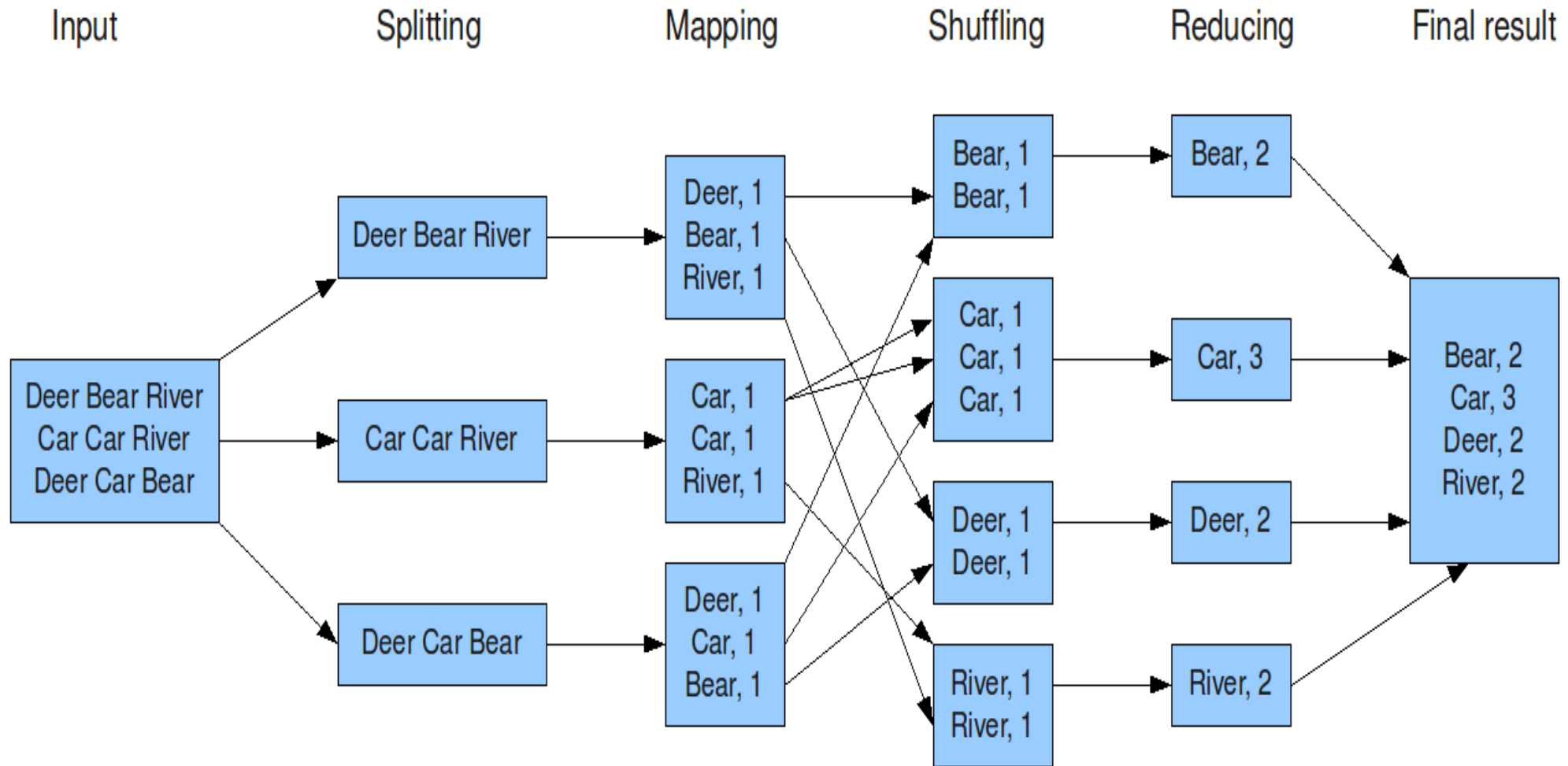
Hadoop Programming with Java

- WordCount
- MaxTemp

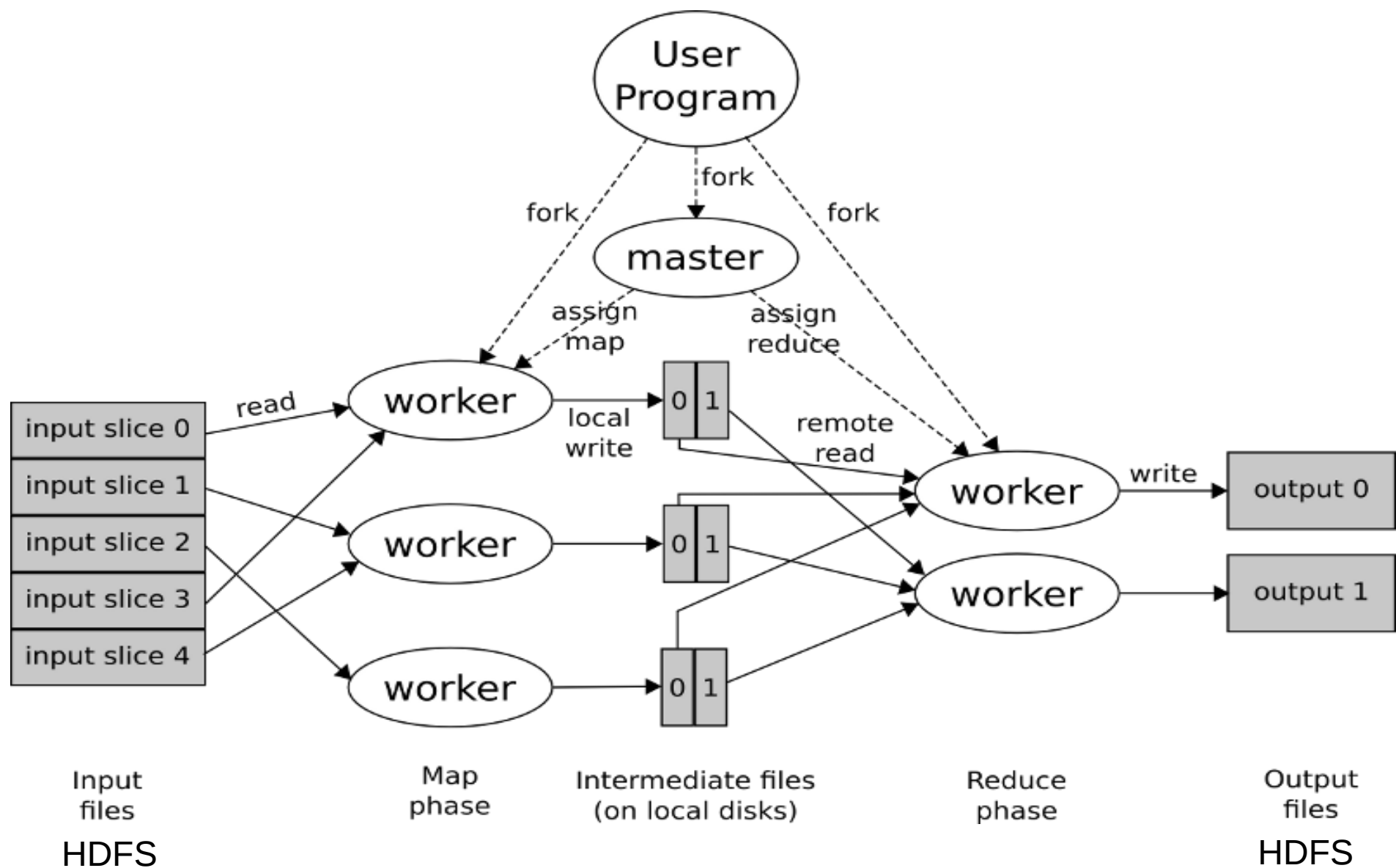
Hadoop Programming with Python

- WordCount
- MaxTemp
- WordLength
- InvertedIndex
- LinkCount

Reminder



Reminder



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 17 physical nodes, of which one is the **MASTER** and the others are **SLAVES**.

All machines are running **CentOS Linux** and they have 16GB of 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

You should see the folder **hadooplab** in your home folder. Check the contents of the folder by executing the command:

ls -las hadooplab

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**

E.g., the following commands are the same:

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

```
hadoop fs -ls
```

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

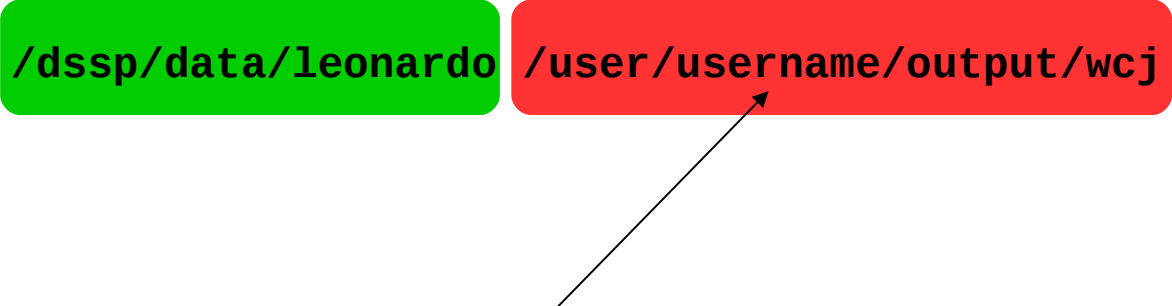
Everything is set! Lets run the job on the cluster.

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
```

MaxTemp: the mapper

```
public class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private static final int MISSING = 9999;

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        String year = line.substring(15, 19);
        int airTemperature;
        if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
            airTemperature = Integer.parseInt(line.substring(88, 92));
        } else {
            airTemperature = Integer.parseInt(line.substring(87, 92));
        }
        String quality = line.substring(92, 93);
        if (airTemperature != MISSING && quality.matches("[01459]")) {
            context.write(new Text(year), new IntWritable(airTemperature));
        }
    }
}
```

MaxTemp: the reducer

```
public class MaxTemperatureReducer extends Reducer<Text, IntWritable, Text,
    IntWritable>{

    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context)
        throws IOException, InterruptedException {

        int maxVal = Integer.MIN_VALUE;
        for (IntWritable value : values) {
            maxVal = Math.max(maxVal, value.get());
        }
        context.write(key, new IntWritable(maxVal));
    }
}
```

MaxTemp: main function

```
public class MaxTemperature {

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MaxTemperature <input path> <output path>");
            System.exit(-1);
        }

        Job job = new Job();
        job.setJarByClass(MaxTemperature.class);
        job.setJobName("Max temperature");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        //job.setNumReduceTasks(2); // 2 reducers

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

MaxTemp: compiling the code

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

```
cd ~/hadooplab/java-maxtemp
```

The relevant code is contained in the files

MaxTemperature.java

MaxTemperatureMapper.java

MaxTemperatureReducer.java

MaxTemp: compiling the code

To compile the code run the command:

```
javac -classpath "$(yarn classpath)" MaxTemperature*.java
```

The file **MaxTemperature.class** must have been produced and also

MaxTemperatureMapper.class

MaxTemperatureRecucer.class

MaxTemp: building the jar

We will create the file

mt.jar

Please execute

jar cf mt.jar MaxTemperature*.class

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

MaxTemp: running the job

Execute the following command:

| | Input dir | Output dir |
|---|---------------------------------|-------------------------|
| <code>hadoop jar mt.jar MaxTemperature</code> | <code>/dssp/data/weather</code> | <code>output/mtj</code> |

MaxTemp: exploring the results

```
hadoop fs -ls output/mtj
```

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
```

MaxTemp: exploring the results

Examine the last lines of the output:

```
hadoop fs -tail output/mtj/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: the reducer

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

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    line = line.strip() # remove leading and trailing whitespace

    word, count = line.split('\t', 1) # parse the input we got from mapper.py

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
            current_count = count
            current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```


WordCount: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.5.1.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-2.5.1.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?

Exercise

Try to run the Maximum Temperature code which can be found in

hadooplab/python-maxtemp

Use as input the hdfs directory

/dssp/data/weather

Use as output the directory

output/mtp

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: the reducer

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

```
from operator import itemgetter
import sys
```

```
current_word = None
current_count = 0.0
counter = 1.0
word = None
```

```
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
        counter = counter + 1
    else:
        current_count = current_count / counter
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
        counter = 1.0
```


WordLength: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.5.1.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: the mapper

```
#!/usr/bin/env python
"""A more advanced Mapper, using Python iterators and generators."""

import sys, os

def read_input(file):
    for line in file:
        # split the line into words
        yield line.split()

def getFileName():
    if 'mapreduce_map_input_file' in os.environ:
        return os.environ['mapreduce_map_input_file']
    else:
        return 'none'

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # 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
        for word in words:
            print '%s%s%s' % (word, separator, getFileName())

if __name__ == "__main__":
    main()
```

Inversion: the reducer

```
#!/usr/bin/env python
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
    # groupby groups multiple word-count pairs by word,
    # and creates an iterator that returns consecutive keys and their group:
    #  current_word - string containing a word (the key)
    #  group - iterator yielding all ["<current_word>", "<count>"] items
    for current_word, group in groupby(data, itemgetter(0)):
        fileList = ""
        prevFileName=""
        for current_word, fileName in group:

            if (fileName != prevFileName):
                fileList = fileList + fileName
                prevFileName = fileName

        print "%s%s%s" % (current_word, separator, fileList)

if __name__ == "__main__":
    main()
```

Inversion: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.5.1.jar  
  -file ./mapper.py -mapper ./mapper.py  
  -file ./reducer.py -reducer ./reducer.py  
  -input /dssp/data/wisconsin  
  -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: 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
```

```
    nodes = line.split()
```

```
    # get the first node
```

```
    v1 = nodes[0]
```

```
    # get the second node
```

```
    v2 = nodes[1]
```

```
    # print key,value pair
```

```
    print '%s\t%s' % (v1, 1)
```

LinkCount: the reducer

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

from operator import itemgetter
import sys

current_node = None
current_count = 0
node = None

for line in sys.stdin:
    line = line.strip()
    node, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    if current_node == node:
        current_count += count
    else:
        if current_node:
            print '%s\t%s' % (current_node, current_count)
        current_count = count
        current_node = node
```

LinkCount: running the job

Write the command in a single line!

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.5.1.jar  
  -file ./mapper.py -mapper ./mapper.py  
  -file ./reducer.py -reducer ./reducer.py  
  -input /dssp/data/wikipedia  
  -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
```