Cloud Networking and Computing
Lab 2: Programming Map-Reduce

Andrzej Duda

1 Introduction to Map-Reduce

In functional languages, `map` is an operation that takes a function `f` mapping a domain D to a range R, and a list `list` of elements of D. It then produces a list of elements in R:

```
map f list

map f [d0, d1, d2...]  -> [f(d0), f(d1), f(d2)...]
```

where [...] denotes a list.

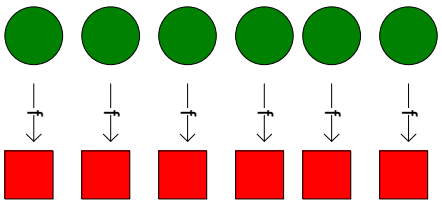

Figure 1: Principle of map.

Examples:

```
map (*2) [1,2,3] = [2,4,6]

square x = x * x
map (square) [1,2,3] = [1,4,9]
```

`reduce` (or `fold`) is an operation that takes a function `f` and a list `list` of elements of D, and continually applies the function to the list. It returns a single value in a range R. It may also take an optional initial value like presented in the figure.

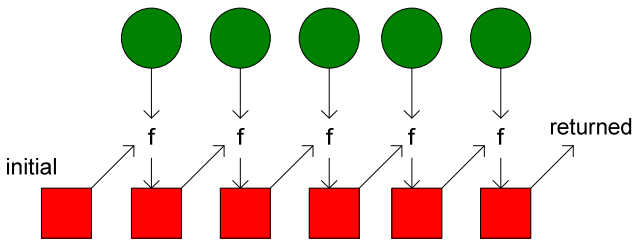

Figure 2: Principle of fold (reduce).

```
reduce f list

reduce f [d0, d1, d2...]  -> r

r = f(f(f(d0, d1), d2), d3...)
```

Examples:

```
reduce (+) [2,4,6] = 12

reduce (*) [2,4,6] = 48

\\ finding the maximum of a list of numerical values
f = lambda a,b:  a if (a > b) else b
reduce (f) [47,11,42,102,13] = 102
```

In the Map-Reduce framework, we define the following functions:

```
map (in_key, in_value) -> [(int_key, intermediate_value)]

reduce (int_key, [intermediate_value]) -> [(out_key, out_value)]
```

Records from the data source (lines out of files, rows of a database, etc.) are *splitted* into parts fed into the `map` function as `(key, value)` pairs: e.g., `(filename, line)`. From the input, `map` produces one or more intermediate values along with an intermediate key.

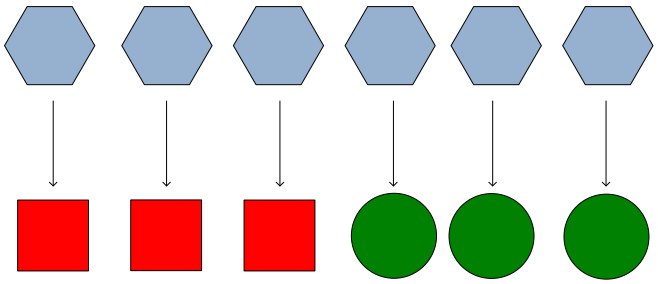

Figure 3: `map` function in Map-Reduce.

After the map phase is over, all the intermediate values for a given intermediate key are combined together into a list. `reduce` combines those intermediate values into one or more final values for that same output key (in practice, usually only one final value per key).

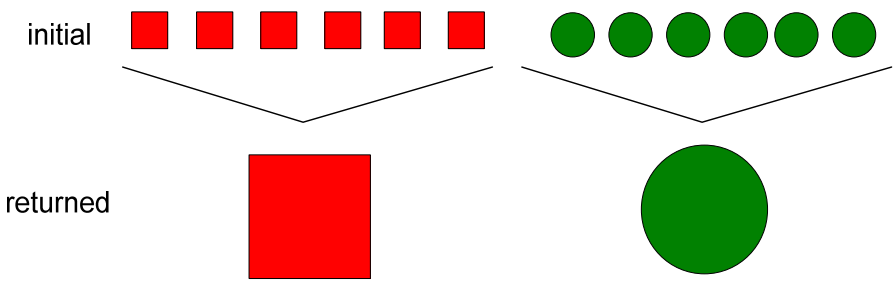

Figure 4: `reduce` function in Map-Reduce.

Figure 4 illustrates the execution in Map-Reduce:

- The initial dataset composed of pairs `(in_key, in_value)`, is *split* into n parts fed into `map` function at n `Mappers`.
- Each `Mapper` produces a list of intermediate results, pairs of `(int_key, intermediate_value)`.
- The results of `map` are *shuffled*: the intermediate values are grouped according to the `int_key` (called output key in the figure). They arrive at each `Reducer` in order, sorted by the key.
- The `Reducer` applies `reduce` function to all values associated with the same `int_key` to generate output key-value pairs.

- The output key-value pairs from each `Reducer` are written persistently back onto the distributed file system (whereas intermediate key-value pairs are transient and not preserved). The output ends up in r files on HDFS, where r is the number of `Reducers`.

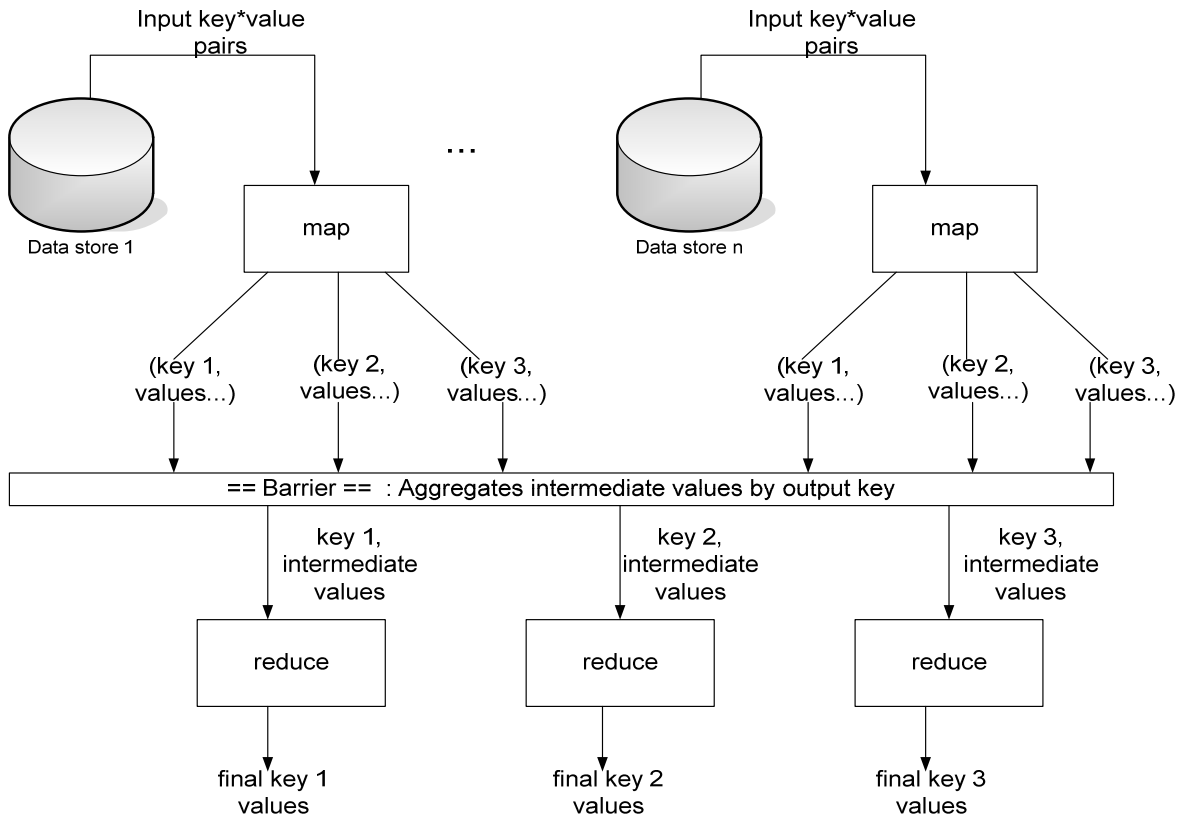

Figure 5: Execution in Map-Reduce.

The number of `Mapper` tasks n is decided based on the number of splits. It is the job of `InputFormat` to create the splits.

For the `InputFormat` based on `FileInputFormat` (an `InputFormat` for handling data stored in files), the splits are created based on the total size, in bytes, of the input files. However, the HDFS block size of the input files is considered as an upper bound for input splits. If you have a file smaller than the HDFS block size, you will have only 1 `Mapper` for that file. If you want to have some different behavior, you can use `mapred.min.split.size` to change.

The number of `Reducer` tasks r is:

$$0.95 * num\_nodes * mapred.tasktracker.tasks.maximum$$

unless the amount of data being processed is small.

2 Getting started

We have prepared an Eclipse project that contains all required libraries. In this way, you can test your programs locally on small files, before running on the cluster, which is very convenient if you need a debugger with breakpoints.

1. Download `TPIntroHadoop.zip` to your workstation/laptop.
2. In the Eclipse menu, click on `File > Import ...`.

3. In the category `General`, select `Existing Projects into Workspace` (and not `Archive`).
4. Choose `select archive file` and find the downloaded archive.
5. Click on `Finish`.

Now, the source directory contains `Question0_0.java`, that can serve as a template for writing each program (but for some questions, please pay attention to changing the types of key/values it declares). Note that `Mapper` and `Reducer` are declared as inner classes: this is not imposed by the framework, it is even not recommended, but we use is to make debugging easier.

2.1 Word count example

This algorithm counts the number of occurrences of every word in a text collection.

```
Class Mapper
Method Map(docid a, doc d)
  for all term t ∈ doc d do
    Emit(term t, count 1)
  end for

Class Reducer
Method Reduce(term t, counts [c1, c2, ...])
  sum ← 0
  for all count c ∈ counts [c1,c2,...] do
    sum ← sum + c
    Emit(term t, count sum)
  end for
```

Figure 6: Word count algorithm

Figure 7 illustrates its operation for a simple input file composed of 2 records. A map task outputs pairs (term, 1), the shuffle phase groups the intermediate results by terms, and the reduce tasks sum the number of occurrences of a given term.

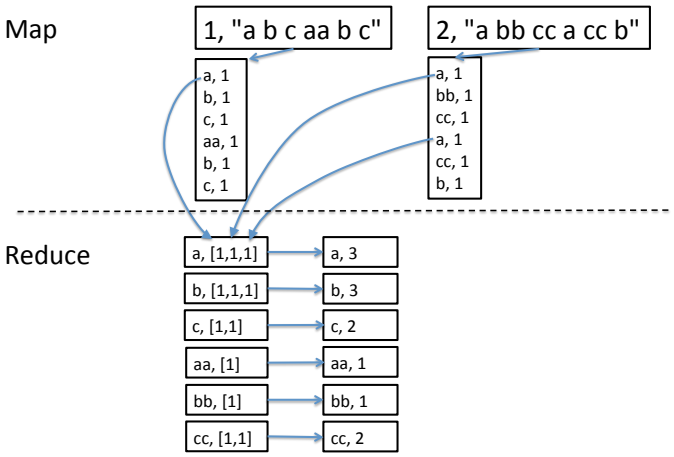

Figure 7: Word count example.

Figure 8 illustrates the phases in the execution of the work count example with another simple input file.

Writing a Map-Reduce program consists of implementing `Mapper` and `Reducer`, defining the data types

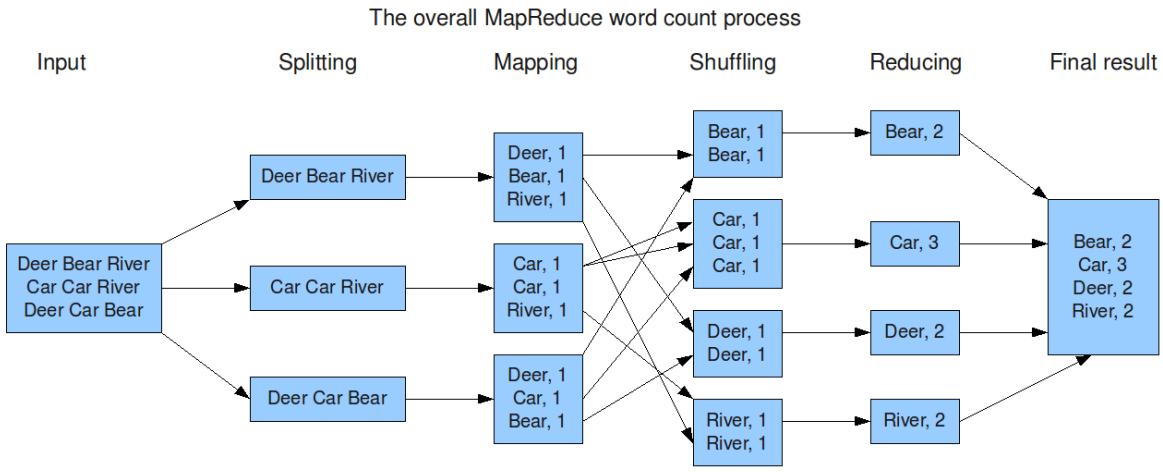

Figure 8: Phases in Map-Reduce.

(keys and values), and serializing methods. Standard data types are provided (`Text, IntWritable, LongWritable` etc.).

Here are the imports:

```
import java.io.IOException ;
import java.util.* ;
import org.apache.hadoop.fs.Path ;
import org.apache.hadoop.io.IntWritable ;
import org.apache.hadoop.io.LongWritable ;
import org.apache.hadoop.io.Text ;
import org.apache.hadoop.mapreduce.Mapper ;
import org.apache.hadoop.mapreduce.Reducer ;
import org.apache.hadoop.mapreduce.JobContext ;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat ;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat ;
import org.apache.hadoop.mapreduce.Job ;
```

Here are the code for the word count `Mapper`:

```
// type input key, type input value, type output key, type output value
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
@Override
   protected void map(LongWritable key, Text value, Context context)
                  throws IOException, InterruptedException {
      for (String word : value.toString().split("\\s+")) {
          context.write(new Text(word), new IntWritable(1));
          }
   }
}
```

and for `Reducer`:

```
// type input key, type input value, type output key, type output value
```

```
public class WordCountReducer extends Reducer<Text, IntWritable, Text, LongWritable> {
@Override
   protected void reduce(Text key, Iterable<IntWritable> values, Context context)
                  throws IOException, InterruptedException {
       long sum = 0;
       for (IntWritable value : values) {
          sum += value.get();
       }
       context.write(key, new LongWritable(sum));
   }
}
```

Here is the main:

```
public class WordCountMain {
    public static void main(String [] args) throws Exception {
         Configuration conf = new Configuration();
         String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
         Job job = Job.getInstance(conf, "word count");
         job.setJarByClass(WordCountMain.class);
         job.setMapOutputKeyClass(Text.class);
         job.setMapOutputValueClass(IntWritable.class);
         job.setOutputKeyClass(Text.class);
         job.setOutputValueClass(LongWritable.class);
         job.setMapperClass(WordCountMapper.class);
         job.setReducerClass(WordCountReducer.class);
         job.setInputFormatClass(TextInputFormat.class);
         job.setOutputFormatClass(TextOutputFormat.class);
         FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
         FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
         System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

2.2 Eclipse cheat sheet

Use the following keyboard shortcuts:

- *Ctrl + Espace* to autocomplete, which can also generate methods (try it in the body of a class)
- *Ctrl + Shift + 1* "Quick fix", to offer relevant actions depending on what is under the cursor.
- *Ctrl + Shift + O* to add missing and remove unnecessary imports.
- *Ctrl + clic* to reach the definition of the name under the cursor (use *Alt* + ← to return to where you were).
- *Alt + Shift + R* to rename what is under the cursor (wherever applicable).

See also `http://eclipse-tools.sourceforge.net/Keyboard_shortcuts_(3.0).pdf`

2.3 Local execution

Before launching a MapReduce program on a cluster, we *always* test it locally on a small sample of the data to process. In this case, the MapReduce framework does not read nor write on HDFS, but on your local file system.

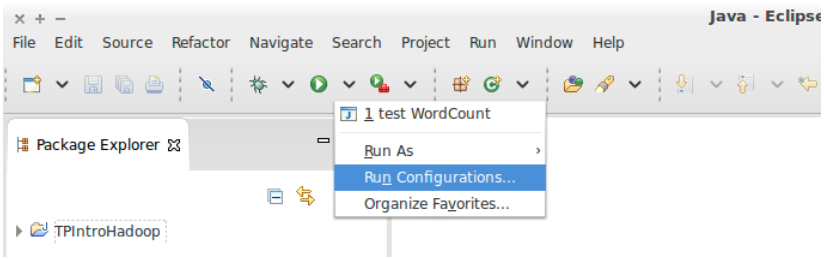

Figure 9: To indicate the settings to use in the local test, you need to create a runtime configuration via the "Run button menu".

Steps to follow:

1. Create a program that counts the number of occurrences of each word in a text file based on the provided code.
2. Create a text file at the project root with the (small) content of your choice.
3. Function `main` takes the path of the input and output file as arguments. So, before launching the program with Eclipse, you need to indicate these paths (you can use the paths relative to the project root), cf. Figure 9.

☞ **1** – Launch the program.

The execution traces appear in the "Console" tab. If there is a bug, you will see an exception at the origin of the crash. Open the generated file and check your results.

When the program is running, the counters maintained by Hadoop appear at the end of execution traces. Add and remove content to/from your text file, restart your program, and observe the evolution of the counters to answer the following questions:

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/duda/Documents/workspace/TPIntroHadoop/lib/slf4j-log4j12-
SLF4J: Found binding in [jar:file:/Users/duda/Documents/workspace/Hadoop1/TPIntroHadoop/lib/slf4j-
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
00:09:50 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... usi
00:09:50 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.sessio
00:09:50 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
00:09:51 WARN mapreduce.JobSubmitter: No job jar file set.  User classes may not be found. See Job
00:09:51 INFO input.FileInputFormat: Total input paths to process : 1
00:09:51 INFO mapreduce.JobSubmitter: number of splits:1
00:09:51 INFO Configuration.deprecation: mapred.job.name is deprecated. Instead, use mapreduce.job
00:09:51 INFO Configuration.deprecation: mapreduce.map.class is deprecated. Instead, use mapreduce
00:09:51 INFO Configuration.deprecation: mapred.input.dir is deprecated. Instead, use mapreduce.in
00:09:51 INFO Configuration.deprecation: mapreduce.reduce.class is deprecated. Instead, use mapred
00:09:51 INFO Configuration.deprecation: mapreduce.inputformat.class is deprecated. Instead, use m
00:09:51 INFO Configuration.deprecation: mapreduce.outputformat.class is deprecated. Instead, use
00:09:51 INFO Configuration.deprecation: mapred.output.value.class is deprecated. Instead, use map
00:09:51 INFO Configuration.deprecation: mapred.output.dir is deprecated. Instead, use mapreduce.o
00:09:51 INFO Configuration.deprecation: mapred.working.dir is deprecated. Instead, use mapreduce.
00:09:51 INFO Configuration.deprecation: mapred.mapoutput.value.class is deprecated. Instead, use
00:09:51 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapreduce.jo
00:09:51 INFO Configuration.deprecation: mapred.mapoutput.key.class is deprecated. Instead, use ma
00:09:51 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.job.user.
```

```
00:09:51 INFO Configuration.deprecation: mapred.output.key.class is deprecated. Instead, use mapre
00:09:51 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1697279223_0001
00:09:51 WARN conf.Configuration: file:/tmp/hadoop-duda/mapred/staging/duda1697279223/.staging/job
00:09:51 WARN conf.Configuration: file:/tmp/hadoop-duda/mapred/staging/duda1697279223/.staging/job
00:09:51 WARN conf.Configuration: file:/tmp/hadoop-duda/mapred/local/localRunner/duda/job_local169
00:09:51 WARN conf.Configuration: file:/tmp/hadoop-duda/mapred/local/localRunner/duda/job_local169
00:09:51 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
00:09:51 INFO mapreduce.Job: Running job: job_local1697279223_0001
00:09:51 INFO mapred.LocalJobRunner: OutputCommitter set in config null
00:09:51 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.Fil
00:09:51 INFO mapred.LocalJobRunner: Waiting for map tasks
00:09:51 INFO mapred.LocalJobRunner: Starting task: attempt_local1697279223_0001_m_000000_0
00:09:51 INFO util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on L
00:09:51 INFO mapred.Task:  Using ResourceCalculatorProcessTree : null
00:09:51 INFO mapred.MapTask: Processing split: file:/Users/duda/Documents/workspace/Hadoop1/file1
00:09:51 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOut
00:09:51 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
00:09:51 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
00:09:51 INFO mapred.MapTask: soft limit at 83886080
00:09:51 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
00:09:51 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
00:09:51 INFO mapred.LocalJobRunner:
00:09:51 INFO mapred.MapTask: Starting flush of map output
00:09:51 INFO mapred.MapTask: Spilling map output
00:09:51 INFO mapred.MapTask: bufstart = 0; bufend = 76; bufvoid = 104857600
00:09:51 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 26214352(104857408); length =
00:09:51 INFO mapred.MapTask: Finished spill 0
00:09:51 INFO mapred.Task: Task:attempt_local1697279223_0001_m_000000_0 is done. And is in the pro
00:09:51 INFO mapred.LocalJobRunner: map
00:09:51 INFO mapred.Task: Task 'attempt_local1697279223_0001_m_000000_0' done.
00:09:51 INFO mapred.LocalJobRunner: Finishing task: attempt_local1697279223_0001_m_000000_0
00:09:51 INFO mapred.LocalJobRunner: Map task executor complete.
00:09:51 INFO util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on L
00:09:51 INFO mapred.Task:  Using ResourceCalculatorProcessTree : null
00:09:51 INFO mapred.Merger: Merging 1 sorted segments
00:09:51 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 98 b
00:09:51 INFO mapred.LocalJobRunner:
00:09:51 INFO Configuration.deprecation: mapred.skip.on is deprecated. Instead, use mapreduce.job.
00:09:51 INFO mapred.Task: Task:attempt_local1697279223_0001_r_000000_0 is done. And is in the pro
00:09:51 INFO mapred.LocalJobRunner:
00:09:51 INFO mapred.Task: Task attempt_local1697279223_0001_r_000000_0 is allowed to commit now
00:09:51 INFO output.FileOutputCommitter: Saved output of task 'attempt_local1697279223_0001_r_000
00:09:51 INFO mapred.LocalJobRunner: reduce > reduce
00:09:51 INFO mapred.Task: Task 'attempt_local1697279223_0001_r_000000_0' done.
00:09:52 INFO mapreduce.Job: Job job_local1697279223_0001 running in uber mode : false
00:09:52 INFO mapreduce.Job:  map 100% reduce 100%
00:09:52 INFO mapreduce.Job: Job job_local1697279223_0001 completed successfully
00:09:52 INFO mapreduce.Job: Counters: 24
File System Counters
FILE: Number of bytes read=508
FILE: Number of bytes written=373263
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
Map-Reduce Framework
Map input records=2
```

```
Map output records=12
Map output bytes=76
Map output materialized bytes=106
Input split bytes=119
Combine input records=0
Combine output records=0
Reduce input groups=6
Reduce shuffle bytes=0
Reduce input records=12
Reduce output records=6
Spilled Records=24
Shuffled Maps =0
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=0
Total committed heap usage (bytes)=514850816
File Input Format Counters
Bytes Read=28
File Output Format Counters
Bytes Written=39

Input file:

a b c aa b c
a bb cc a cc b

Output file (part-r-00000):

a 3
aa 1
b 3
bb 1
c 2
cc 2
```

Question: **(2 point)** What do the counters `Map input records` and `Map output records` represent?
Answer: "records" are couples $(key, value)$. These two counters thus indicate the number of couples on in (2) and on out of the *map* function (12). On input, it is the number of lines read from the input file. On output, it is the number of words of the document.

Question: **(1 point)** What is the link between `Map output records` and `Reduce input records`?
Answer: They are identical: the data generated by *map* are directly transmitted by the framework, to the *reduce* functions.

Question: **(2 point)** What does the counter `Reduce input groups` represent?
Answer: The intermediate pairs between *map* and *reduce* are grouped by keys. The number of groups is thus the number of distinct keys (in our example, the words - 6), which corresponds to the number of times the framework will call the *reduce* function.

3 Execution on the cluster

To work on the cluster:

☞ **2** – Open a shell on delos with `ssh liku@delos.imag.fr`.

☞ **3** – Open a shell on the master node with `ssh duda@152.77.78.100`.

The program written in the previous part should work on the cluster without changes, if it is running locally. But we must first make a compiled package:

- In Eclipse, right click on the `src` directory. Choose "Export ..."
- Open the "Java" category and double-click on "JAR file".
- Check if the directory `lib` is **not selected** in the resources to export,
- Enter the package name, e.g. `tp-group1.jar`.

The package will be generated in the root of your Eclipse workspace. Transfer it with `scp` to the home directory on the master node `152.77.78.100`.

We now count the number of occurrences of each word in the 5 volumes of *Misérables* by Victor Hugo. If your main class is called `Question1_1`, run your program on the master node by typing:

☞ **4** – `hadoop jar tp.jar Question1_1 /data/miserables wordcount`

The results will be stored in the `wordcount` directory.

```
duda@NameNode:~$ hadoop jar tphQ1.jar Question1_1 /data/miserables wordcount
16/04/26 06:10:19 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platfo
16/04/26 06:10:20 INFO client.RMProxy: Connecting to ResourceManager at NameNode/152.77.78.100:803
16/04/26 06:10:20 INFO input.FileInputFormat: Total input paths to process : 5
16/04/26 06:10:21 INFO mapreduce.JobSubmitter: number of splits:5
16/04/26 06:10:21 INFO Configuration.deprecation: user.name is deprecated. Instead, use mapreduce.
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapreduce
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.output.value.class is deprecated. Instead
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.mapoutput.value.class is deprecated. Inst
16/04/26 06:10:21 INFO Configuration.deprecation: mapreduce.map.class is deprecated. Instead, use
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.job.name is deprecated. Instead, use mapr
16/04/26 06:10:21 INFO Configuration.deprecation: mapreduce.reduce.class is deprecated. Instead, u
16/04/26 06:10:21 INFO Configuration.deprecation: mapreduce.inputformat.class is deprecated. Instea
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.input.dir is deprecated. Instead, use mapr
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.output.dir is deprecated. Instead, use ma
16/04/26 06:10:21 INFO Configuration.deprecation: mapreduce.outputformat.class is deprecated. Inst
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use mapr
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.output.key.class is deprecated. Instead,
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.mapoutput.key.class is deprecated. Instead
16/04/26 06:10:21 INFO Configuration.deprecation: mapred.working.dir is deprecated. Instead, use ma
16/04/26 06:10:21 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1444991602904_0775
16/04/26 06:10:21 INFO impl.YarnClientImpl: Submitted application application_1444991602904_0775 t
16/04/26 06:10:21 INFO mapreduce.Job: The url to track the job: http://152.77.78.100:8089/proxy/ap
16/04/26 06:10:21 INFO mapreduce.Job: Running job: job_1444991602904_0775
16/04/26 06:10:28 INFO mapreduce.Job: Job job_1444991602904_0775 running in uber mode : false
16/04/26 06:10:28 INFO mapreduce.Job:  map 0% reduce 0%
16/04/26 06:10:35 INFO mapreduce.Job:  map 20% reduce 0%
16/04/26 06:10:36 INFO mapreduce.Job:  map 40% reduce 0%
16/04/26 06:10:37 INFO mapreduce.Job:  map 60% reduce 0%
16/04/26 06:10:40 INFO mapreduce.Job:  map 100% reduce 0%
```

```
16/04/26 06:10:43 INFO mapreduce.Job:  map 100% reduce 100%
16/04/26 06:10:43 INFO mapreduce.Job: Job job_1444991602904_0775 completed successfully
16/04/26 06:10:44 INFO mapreduce.Job: Counters: 44
File System Counters
FILE: Number of bytes read=5063675
FILE: Number of bytes written=10608973
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=2536148
HDFS: Number of bytes written=623905
HDFS: Number of read operations=18
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=5
Launched reduce tasks=1
Data-local map tasks=4
Rack-local map tasks=1
Total time spent by all maps in occupied slots (ms)=315680
Total time spent by all reduces in occupied slots (ms)=42704
Map-Reduce Framework
Map input records=52711
Map output records=421739
Map output bytes=4220191
Map output materialized bytes=5063699
Input split bytes=610
Combine input records=0
Combine output records=0
Reduce input groups=52555
Reduce shuffle bytes=5063699
Reduce input records=421739
Reduce output records=52555
Spilled Records=843478
Shuffled Maps =5
Failed Shuffles=0
Merged Map outputs=5
GC time elapsed (ms)=207
CPU time spent (ms)=13490
Physical memory (bytes) snapshot=1641422848
Virtual memory (bytes) snapshot=7386574848
Total committed heap usage (bytes)=1226178560
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=2535538
File Output Format Counters
Bytes Written=623905

real 0m25.643s
user 0m4.464s
```

```
sys 0m0.244s

duda@NameNode:~$ hdfs dfs -cat wordcount/part-r-00000|more

12379
"Defects," 4
"Hu!" 1
"Information 4
"Le 1
"Mon 1
"Plain 8
"Project 20
"Qu'est-ce 1
"Right 4
"Voilà 1
"bien 1
"chercher 1
"cheval 1
"et 1
"recettes" 1
"remise 1
"vierges" 1
#17489] 1
#17493] 1
#17494] 1
#17518] 1
#17519] 1
$5,000) 4
'AS-IS',4
("the 4
($1 4
(1861),1
(1862) 5
(801) 4
(Habana)_. 1
(Il 2
(Je 1
(Matth.,4
(Mlle 9
(Seine-et-Oise). 1
(_Ai-je_ 1
(_Parbleu_,1
(_ici 1
(_la 1
(_sic_) 1
(_six 1
(a) 4
(académie 1
(and 4
(any 4
(au 1
(available 4
.....
```

Question: **(1 point)** In the traces at the beginning of the execution, find the number of *splits* read on HDFS. Which counter does this number correspond to?
Answer:

```
Launched map tasks. The framework creates one instance of Mapper per input split
- 5.
```

■