Wordcount example: RHadoop from the command line

A task which can be done in iterations (like finding the item in a maze) is the kind of task computers excel at, while humans "manually" doing it, is very inefficient.

© Shutterstock

So far you have managed to run `Hadoop` and `R`, connect `R` with `Hadoop` via `RHadoop` and learned about the `R` libraries that are used for working with the `map` and `reduce` tasks. In this article we will show how to write and example using `R` and `Hadoop` from the command line.

For demonstration purposes, we will use the same standard example of finding the first ten most frequent occurrences of words in a file using `Rhadoop`. To do that, apart from the data set, you will also need to save the scripts for the map-and-reduce tasks when you get to Steps 2.1 and 2.2 below.

Before we start with the eaxmple we need to start `Hadoop` by typing in the terminal window the following commands (see the instructions from the beginning of this week):

```
start-dfs.sh
start-yarn.sh
```

The procedure for solving the example in `R` consists of five steps, as follows.

Step 1

Go to the folder /home/hduser/week2 where you have saved the file Term_frequencies_sentence-level_lemmatized_utf8.csv. In this step, you have to save the

Term_frequencies_sentence-level_lemmatized_utf8.csv under the name testR.csv. You can do that using the cp command:

```
hduser@osboxes ~/week2 $ cp Term_frequencies_sentence-level_lemmatized_utf8.csv testR.csv
```

We renamed the file so that it has a shorter name for easier writing. Next, copy the file from your local file system to the Hadoop file system using the command copyFromLocal:

```
$ hadoop fs -copyFromLocal ~/week2/testR.csv
.
```

Note that you might have to modify the last command with the path where the file testR.csv resides in the local filesystem.

To check whether the file has been copied write:

```
$ hadoop fs -ls
```

The result should show the existence of the file testR.csv in the Hadoop filesystem.

Note that this is a very important step. The file testR.csv contains data that are handled by Hadoop. Data reside somewhere in the Hadoop filesystem; therefore, it is not enough to save these data only in your local folder, as is the case with the R scripts for map and reduce.

Step 2

Save the R scripts for map and reduce on your local machine using the nano editor. Open the nano editor using the command:

```
$nano
```

Step 2.1 Map script

Then copy the following script for Map and paste it into the editor.
The source code for the map task should be saved in a map.R file and is as follows:

```
map_wc <- function(.,lines)
{

  lines_lst = unlist(strsplit(lines,"\r\n",fixed=TRUE))
  l_cnt<-1;
  keys_l<-c()
  data_l<-c()
  for (line in lines_lst)
  {
    words = unlist(strsplit(line,";",fixed=TRUE))

    if (length(words) != 5){
      next
    }


    for (i in 2:4)
    {
      keys_l[l_cnt] = i-1
      x = as.numeric(words[i])
      y = words[1]
      data_l[[l_cnt]] = c(x,y)
      l_cnt <- l_cnt + 1
    }
  }
  return(keyval(keys_l, matrix(unlist(data_l), ncol = 2, byrow = TRUE)))
}
```

Copy the script (ctrl+c) and paste and save (ctrl+shift+v) it to the nano editor under the name map.R in the /home/hduser/week2 folder.

Let us describe what this script does.

The name of the function that performs the mapping in R is called map_wc. This function has two input arguments, a key and a value. The map function takes two arguments, a key and a value. The key here is not important, indeed it is always NULL, hence the dot in the map_wc <- function(.,lines). The second argument, the value, is called “lines”. The value here contains several lines of text, which are split according to some rule

```
map_wc <- function(.,lines)
```

The first part of the code:

```
  lines_lst = unlist(strsplit(lines,"\r\n",fixed=TRUE))
  l_cnt<-1;
  keys_l<-c()
  data_l<-c()
  for (line in lines_lst)
  {
    words = unlist(strsplit(line,";",fixed=TRUE))

    if (length(words) != 5){
      next
    }
```

is responsible for reading the data set in R (for example testR.csv file), and stripping it in a format suitable for the following for loop. The next for loop:

```
for (i in 2:4)
    {
      keys_l[l_cnt] = i-1
      x =
as.numeric(words[i])
      y = words[1]
      data_l[[l_cnt]] =
c(x,y)
      l_cnt <- l_cnt + 1
    }
```

is responsible for preparing the (key, value) pairs.

Step 2.2 Reduce script

Repeat the same procedure for reduce and save it under the name reduce.R.

In R, in the example of finding the ten most frequent words in a file, the reducer takes all the values that have the same key, for example key2 == 1, sorts them in decreasing order and lists the first ten maximum values.

Next, save the following source code for the reducer under the name reduce.R:

```
reduce_wc <- function(k,v)
{
  srt = sort(as.numeric(v[,1]), decreasing=TRUE, index.return=T)
  NF = 10
  keyval(k,list(v[srt$ix[1:NF],]) )
}
```

In the first row, the reduce function called “reduce_wc” is defined with two arguments: “k” and “v”, where “k” stands for “key” and “v” stands for “value”.

```
reduce_wc <- function(k,v)
```

Next, a sort function is called. The sort function has the following syntax:

```
sort(x, decreasing = FALSE, index.return,
...)
```

where

- x: is the data that are sorted
- decreasing: defines the order of the sorting. In our example this value is set to TRUE so that we obtain the values sorted in decreasing order
- index.return: defines whether the ordering index vector should be returned. In our case it is set to TRUE. We need this value so that we can list the keys and their values in the final command:

```
keyval(k,list(v[srt$ix[1:NF],])
)
```

Note here that the names of the files `map.R` and `reduce.R` are different from the names of the functions `map_wc` and `reduce_wc`, which perform the map and reduce tasks. This is important because in Step 4 we will make calls to the functions `map_wc` and `reduce_wc`, and not the filenames `map.R` and `reduce.R`.

Step 3

Make sure that you are positioned in the same folder, where you have saved `map.R` and `reduce.R`, for example in /home/hduser/week2. Next, to run `R` in Linux we simply write:

```
$ R
```

In this step you have to initialize `RHadoop`, as explained in the RHadoop initialization. For the initialization use the following set of commands:

```
Sys.setenv(HADOOP_OPTS="-Djava.library.path=/usr/local/hadoop/lib/native")
Sys.setenv(HADOOP_HOME="/usr/local/hadoop")
Sys.setenv(HADOOP_CMD="/usr/local/hadoop/bin/hadoop")
Sys.setenv(HADOOP_STREAMING="/usr/local/hadoop/share/hadoop/tools/lib/hadoop-
streaming-2.6.5.jar")
Sys.setenv(JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64")

library(rmr2)
library(rhdfs)
hdfs.init()
```

Copy and run each of them into the command line.

Next, load the saved `R` scripts for `map and reduce` using the command “source(path-name-to-script)” as follows:

```
source('/home/hduser/week2/map.R')
source('/home/hduser/week2/reduce.R')
```

Do not forget to change the path to the `map.R` and `reduce.R` in case you have saved them in a different folder.

Step 4

To start MapReduce in R use the following code

```
rr<-from.dfs(mapreduce(input='testR.csv', input.format='text', map = map_wc,
reduce=reduce_wc, combine=F))
```

In the last command we do the following:

- `from.dfs`: this call requests results directly from dfs into `R` variable that we named `rr`,
- input: here we provide the path to the filename in which we have saved the data in hadoop dfs. In our case the filename is `testR.csv`. Note that you might have to modify the name `testR.csv` with the current path in the dfs,

- input.format: here we define the data format. In our case the data are given in text format,
- map: here we define the name of the map function. In our case it is `map_wc` (as defined in `map.R`in Step 2),
- reduce: here we define the name of the reduce function. In our case it is `reduce_wc` (as defined in `reduce.R` in step 2)
- combine: here we define whether `combine` is required. The value “F” stands for “False”, meaning that in our example we do not perform `combine`.

As a result of the last command we get three lists of words that contain the 10 most frequent words used in three context meanings: positive meaning, neutral meaning or negative meaning.

Step 5

To list them - wait for a while… And get the results in rr. Hit:

```
rr <enter>
> rr
$key
[1] 1 2 3

$val
$val[[1]]
      [,1]    [,2]
 [1,] "74061" "biti"
 [2,] "25816" "v"
 [3,] "17393" "za"
 [4,] "16687" "in"
 [5,] "14790" "na"
 [6,] "12745" "da"
 [7,] "12722" "se"
 [8,] "9816"  "z"
 [9,] "9358"  "pa"
[10,] "8610"  "ki"

$val[[2]]
      [,1]     [,2]
 [1,] "145951"
"biti"
 [2,] "51246"  "v"
 [3,] "36222"  "in"
 [4,] "35603"  "za"
 [5,] "30307"  "na"
 [6,] "27220"  "da"
 [7,] "23987"  "se"
 [8,] "22683"  "z"
 [9,] "18630"  "pa"
[10,] "18320"  "ki"

$val[[3]]
      [,1]    [,2]
 [1,] "37957" "biti"
 [2,] "14936" "v"
 [3,] "11971" "in"
 [4,] "10659" "za"
 [5,] "9303"  "na"
 [6,] "7428"  "se"
 [7,] "6633"  "z"
 [8,] "5922"  "da"
 [9,] "4984"  "ki"
[10,] "4764"  "pa"
>
```

And you are done. To close the `RHadoop` environment correctly please type:

```
q()
stop-yarn.sh
stop-dfs.sh
```

Note that you will be asked after typing `q()` whether to save workspace image or not. We suggest to enter `n`.