LAB 2 - WEEK 3

MAPREDUCE WORD COUNT LAB – FOR DOCKER

This document is divided into two parts.

1. Practice Lab

- MapReduce Java WordCount Implementation

  Just try to run through all the steps and see if they work properly for you. It's very essential to get the word count program run properly on your machine.

No need to submit this part.

2. Homework Questions

a. Do some research and find out what code needs to be added to the Word Count program for automatic removal of "output" directory before job execution.

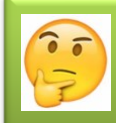
b. Run the above basic Word Count program in pseudo-distributed mode with 2 reducers. Use `setNumReduceTask` method of `Job` object. Paste the screenshot of the two `part-r-*` files created in HDFS.

c. Modify the *WordCount* program to output the counts of only the words "Hadoop" and "Java". (This is case-insensitive count. For example, *"Hadoop"* and *"hadoop" should be counted* as same word!)
Submit your output file along with the java program.

d. Modify the WordCount program to output the counts of only those words which appear in the document at least 25 times. (Use case-insensitivity)
Submit your output file along with java program.

e. Write a MapReduce program to find out how many distinct (unique) words are there in the input file. (Use case-insensitivity)
(Hint: Recall that there are setup and cleanup methods in the Reducer class)
Submit your output file along with the java program.

In your lab submission, I should be able to find the java programs for all (a), (b), (c), (d) and (e) with commands to run these programs in pseudo-distributed mode.
For (b), I'll need a screenshot as mentioned.



Food for thought

What if you want to produce an output file which will be sorted on Word Counts and not on Words?

Practice Lab - MapReduce Java WordCount Implementation

The purpose of this practice lab is to give you a feel of running MapReduce programs in Hadoop pseudo-distributed environment on Docker.

Make sure that you can run the given java Word Count program both locally and in pseudo-distributed mode.

1. Get the MR Word Count project by cloning the github repository:
   `git clone https://github.com/himrudula/cs523bdt-MR-WC-Lab.git`
2. You'll see a new folder named ***cs523bdt-MR-WC-Lab*** now. Open this project folder in your favorite IDE.
3. Build the project by doing Maven Install.
   `mvn clean install`

3. Hadoop Local Mode Setup for Windows

For running MapReduce programs in local mode, you need to set up a local Hadoop "Home" on your machine.

1. Create a folder on your machine, for example: C:\hadoop
2. Inside it, create a bin folder: C:\hadoop\bin
3. Hadoop requires native Windows libraries (specifically hadoop.dll and winutils.exe) to interact with the file system, but the standard Apache Hadoop distribution does not include them. I've made these files available for you in Lab1 instructions. Put the hadoop.dll and winutils.exe files you've received earlier into C:\hadoop\bin
4. Configure Environment Variables:

   Set HADOOP_HOME:

   a. Create a new System Environment Variable named HADOOP_HOME.

   b. Set the value to your parent folder: C:\hadoop (Do not include \bin here).

   Update Path:

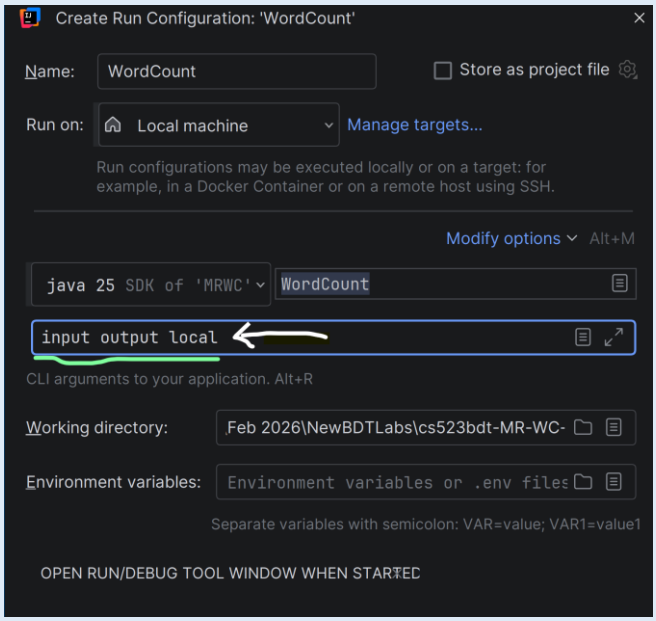
   a. Edit your System Path variable.

   b. Add a new entry: %HADOOP_HOME%\bin

Run WordCount in Local Mode:

- Notice the "input" folder in your project directory. It should've the input file "Lab2-WC-Input.txt" there.
- You'll need to supply runtime arguments to your program as "input", "output", "local". These are the folder names which Hadoop will use to take input from and store output to, respectively. (alternatively, you can choose to have your own names!)

  Below is the screenshot of IntelliJ which shows how to give runtime arguments.



- Run the Java program now and see that the output directory got created in your project path and there's the output file named "part-r-00000" which has the counts of all the words from your input file.

Run WordCount in Pseudo-distributed Mode:

- Run **`mvn package`** command to create a jar file of your program. You can find this jar file inside the "target" directory in your project folder.
- Save this jar file inside the "my_code" folder that you've created during Lab1 setup. Also, copy the input file *Lab2-WC-Input.txt* into this "my_code" folder.
- Now, let's create "input" folder in HDFS. For this, make sure that the **`cs523bdt-lab`** docker container is running. Then use the following command to access the shell of the container:

  `docker exec -it cs523bdt-lab bash`

  Now, under /user, let's create input folder:

  `hadoop fs -mkdir /user/input`

```
[2026-02-22 22:04] root@localhost /opt $ hadoop fs -ls /user
Found 3 items
drwxr-xr-x   - root supergroup          0 2026-02-13 16:59 /user/hive
drwxr-xr-x   - root supergroup          0 2026-02-22 22:03 /user/input
drwxr-xr-x   - root supergroup          0 2026-02-13 16:59 /user/root
[2026-02-22 22:04] root@localhost /opt $
```

- Put the given input file in this newly created HDFS input folder.

  `hadoop fs -put my_code/Lab2-WC-Input.txt /user/input`

- Run your MR word count job using the command given below.

```
hadoop jar my_code/MRWC-1.0.jar WordCount /user/input /user/output
```

- After running the program, check the output part-r file in HDFS. You can do cat on this file to see it's contents.

```
[2026-02-22 22:18] root@localhost /opt $ hadoop fs -ls /user/output
Found 2 items
-rw-r--r--   1 root supergroup          0 2026-02-22 22:18 /user/output/_SUCCESS
-rw-r--r--   1 root supergroup       3159 2026-02-22 22:18 /user/output/part-r-00000
[2026-02-22 22:20] root@localhost /opt $
```

- You can also see the part-r file using HDFS file browser: