|  |  |
| --- | --- |
|  | |
| Programming for Big Data –Assignment 1  Hadoop/Map-Reduce | |
|  | |
| Ciaran Finnegan – TU060 (Part Time) – Second Year  PROG9813: 2022/2023  MSc in Computer Science (Data Science)  Student No : D21124026  26/2/2023 |  |
|  |  |

Table of Contents

1 Introduction 3

1.1 Purpose of Report 3

2 Loading Data Into HDFS 4

2.1 Setting Up Shared Folders 4

2.2 Language Files Source (Books) 4

2.3 Transferring and Loading Data Files 5

3 Map-Reduce Process Design 7

3.1 High Level Overview 7

3.2 Explanation of Map-Reduce Workflow 8

3.3 Key Design Decisions and Assumptions 15

4 Map-Reduce: Java Code 17

4.1 Driver 17

4.2 Job 1 20

4.3 Job 2 23

5 Python: Graph Analysis of MR Outputs 26

5.1 Set Up Excel From HSFs Output 26

5.2 Python Jupyter Notebook 27

5.3 Analysis Commentary 33

# Introduction

## Purpose of Report

This report collates all the delivery requirements for Assignment 1 in the Programming for Big Data module (PROG9813: 2022-23).

The primary content of the report covers;

* How language data (book files) were sourced and then loaded into a HDFS data store on a Linux VM running Hadoop (the VM provided during lecture 1).
* How the Map-Reduce workflow is constructed to ingest the language files stored in HDFS and ultimately compile a breakdown of the average character frequency in each language.
* Design assumptions and decisions made during the implementation of this PBD assignment.
* The Java source written to build the Map-Reduce processes, and how the features offered by Map-Reduce were employed to solve the assignment challenge.
* Python source code and graphs (via a Jupyter Notebook) of the analysis on the output from the Map-Reduce process.

Supporting source code and HDFS outputs accompany this report as part of the overall Brightspace submission.

# Loading Data Into HDFS

## Setting Up Shared Folders

The VM provided during lectures was used as the Hadoop environment for this assignment. Following lecturer advice, a shared folder on the host machine was set up.

Graphical user interface, text, application

Description automatically generated

*Figure – VM Shared Folder Set Up*

## Language Files Source (Books)

Six books were selected from the <http://www.gutenberg.org> website;

* Two in Spanish.
* Two in German.
* Two in Italian.

Graphical user interface

Description automatically generated with low confidence

*Figure – Book Files Extracted From Gutenberg Site*

## Transferring and Loading Data Files

The filenames of each book were altered in the transfer process from host machine to VM to add a suffix indicating language.Graphical user interface, text

Description automatically generated

*Figure – Shell Script with Linux Commands to Copy/Rename Files in VM*

The renaming was done as part of the design assumptions covered in section 3.3 of this report. Setting up a book directory and loading the files into HDFS was done through the following Linux commands;

Graphical user interface, text, application

Description automatically generated

*Figure – Shell Script with Linux Commands to Copy Files to HDFS*

The Hadoop interface confirmed that the language book files were successfully loaded.Graphical user interface, table

Description automatically generated with medium confidence

*Figure –HDFS Directory Populated with Book Files*

A quick inspection (example below) showed that the data was intact and ready for processing.Graphical user interface, text, application, email

Description automatically generated

*Figure – Example of a German Book File ‘Chunk’ Stored in HDFS*

# Map-Reduce Process Design

## High Level Overview

This diagram displays the high-level Map-Reduce process built and deployed for the assignment.

Diagram

Description automatically generated

*Figure – Map-Reduce Process Overview to Calculate Language Letter Frequency*

The following sections of this report elaborate on the individual Map/Reduce processes.

## Explanation of Map-Reduce Workflow

The Java code written for this assignment, compiled in Eclipse under a jar file called AverageLetterFrequency, is provided in Section 4 of this report. This section (Section 3.2) provides a generalised overview of the steps in the assignment Map-Reduce process.

### Driver Process

There is one Driver process for the entire assignment. Key steps in the process are;

* Invoke jar with command line parameters. This is the main Java project class programme. The AverageLetterFrequency jar file is invoked with two parameters.
  + The first is the location of the language files in the HDFS repository.
  + The second is the output directory to be created in HDFS to store the output from the (job1) Map-Reduce process.

Graphical user interface, text

Description automatically generated

*Figure – Cmd Line Parameters to Invoke Jar file for Map-Reduce Assignment*

* Use Wildcards to select a subset of language specific book files stored in HDFS (see figure above). This glob is used to set the input path of the first job.
* File name/format is used to indicate language. All files processed by the Map-Reduce process must be in the file name format: ***<bookname>\_<language>***. The MR process will read in all files with the given language name used in the command line parameter invoking the jar file (see figure above).
* Mapper and Reducer input and output formats are set up as a Key Value pair of lists of <Text, Integer> datatypes.
* Set up two jobs in the driver class:
* Job 1 reads all the selected language files in the HDFS repository. The Job 1 output to HDFS is a count for the total number of appearances for each individual letter character in the books for the selected languages.
* Job 2 takes the output from Job 1 as an input. Character totals are recreated. The total count of ALL characters in the books is also created in the Job2 MR process. This total of all characters is used to calculate the frequency of each individual language character. The Job 2 Reducer output contains the frequency values for all characters, which is stored in a file in the newly created ***CA1\_Output\_out*** folder.
* Job1 sets up a Combiner and Partitioner Hadoop process, which are explained in the following sections.
* Job 2 only uses a Mapper and a Reducer process.
* A job sequence is set up so that Job 1 must complete first, before Job 2 is permitted to commence.

### Job 1: Mapper/Combiner Process

Key steps in the first (Job1) Mapper process are;

* The function AverageLetterMapper is set up to process HDFS input line by line from the specific language files.
* Each line is split into individual characters.
* Non-alpha characters are excluded.
* A regex expression is used to ensure local language characters are included, along with English language characters .
* Counts of characters per line are stored in a Java HashMap and then extracted to write out individual <Char, Count> Key:Value pairs from the Mapper.
* A Combiner class is set up in the Driver class using the same code as the Mapper (job1.setCombinerClass(AverageLetterReducer.**class**);). This acts as a mini-Reducer running locally on the Mapper output and generates the same output format as the Mapper. It is used here to improve Job1 performance.

..

### Job 1: Partitioner Process

Key steps in the Partitioner process are;

* A Partitioner class is set up in the Driver code (job1.setPartitionerClass(AverageLetterPartitioner.**class**);).
* The number of Reducer tasks for Job 1 is explicitly set to ‘2’ (job1.setNumReduceTasks(2);).
* The choice of Reducer to be selected is determined in the Partitioner process. If a character is a vowel the Reducer processing is directed to Reducer 1. Non vowels are directed to Reducer 2.
* The reason for this partition logic is that there are more occurrences of vowels than non-vowels in a language text. However, the number of vowels is much less (5), therefore this split is used in this MR process distribute effort across the Reducers.

### Job 1: Reducer Process

Key steps in the first (Job1) Reducer processes are;

* The function AverageLetterReducer is set to read the inputs from the Mapper process, after Hadoop sorting and grouping has taken place.

* Key values, which are individual language characters, are ingested by the Job1 Reducer processes with a list of integer counts for character occurrences in each line of the input.
* The Reducers sum all the character integer counts for each character (Key).
* The output of *<character, Total Count>* is written to HDFS. The two Reducer jobs write to separate files in the ***CA\_Output*** directory, which was set in the command line to execute the assignment jar file.

### Job 1: MR Process Outputs

Job 1 output is stored under in the ***CA\_Output*** directory, created based on the configuration set up in the Driver class.

Graphical user interface

Description automatically generated

*Figure – CA1\_Output directory – Job 1 Output*

The ‘***part-r-0000***’ file contains the total count of occurrences for vowel characters read in the language file inputs.

Graphical user interface, text, application, email

Description automatically generated

*Figure – CA1\_Output directory – Job 1 Output – Reducer 1 – Vowels*

The ‘***part-r-0001***’ file contains the total count of occurrences for non-vowel characters read in the language file inputs (sample represented below).

**Graphical user interface, application

Description automatically generated**

*Figure – CA1\_Output directory – Job 1 Output – Reducer 2 – Non-Vowels*

### Job 2: Mapper Process

Key steps in the second (Job2) Mapper process are;

* The FreqDistrbMapper class reads the output from Job 1, which is stored in the **CA1\_Output** directory in HSFS.
* The input for this Mapper process is much smaller than in Job 1. The Key values in the **CA1\_Output** directory are unique, so the Mapper effectively repeats the output process of Job1.
* However, a static key of ***‘Total\_Chars’*** is also written out by the Mapper as each individual character is re-output with the number of total occurrences. The use of this static key means that it generates a master total output that grows as each separate character is read/output.
* In addition, a ‘***Local\_Char’*** static key is set up to group all non-English language characters. This is done to collate characters such as *ä, ö,* and *ü* (for example) into a single Key category. The purpose of this grouping is to simplify the end result comparison graphs in the Python output for this project.

### Job 2: Reducer Process

Key steps in the second (Job2) Reducer process are;

* The *reduce* function in the FreqDistrbReducer class performs a simple summation of the values in the iterable Integer list for each individual key. There would be no real sorting or grouping required from Hadoop.
* This summation includes the value for language specific characters such as *ä, ö,* and *ü* (for example), which are grouped under a single Key vale named ‘***Local\_Char’.***
* A static Key called ***‘Total\_Chars’*** is fed into the Reducer and this summation of the Integer list for this Key will generate the value representing the **total** number of characters in the language book file inputs.
* The *reduce* function does not call the *context.write* routine, as configured by the Driver class. Instead, it incrementally fills a Java HashMap with each Key/Value pair, including the value for the count of all characters.
* The Job 2 Reducer has a *cleanup* routine that executes after main processing is complete. This cycles through the Java HashMap created by the *reduce* function and generates a distribution value for each character, based on a division of character totals by total numbers of characters.
* The Key value for final input is concatenated with the name of the language, which has been parsed from the command line input into a Job 2 configuration custom variable called ‘*language.text*’.
* The *cleanup* routine output is written to a HSFS directory called ***CA1\_output\_out***.

### Job 2: MR Process Outputs

Job 2 output is stored under in the ***CA\_Output\_out*** directory, created based on the configuration set up in the Driver class.

Table

Description automatically generated

*Figure – CA1\_Output\_out directory – Job 2 Output*

Job 2 output is one file.

Graphical user interface, text, application, email

Description automatically generated

*Figure – CA1\_Output\_out directory – Job 2 Output File*

The ‘***part-r-0000***’ file contains the final breakdown of average letter frequencies and the description of the language (German example represented below).

Graphical user interface, application

Description automatically generated

*Figure – CA1\_Output\_out directory – Job 2 Final Output*

To maintain the Integer data type for the Job 2 Reducer output, the average letter frequency is multiplied by 10000, The display value is adjusted to a > 1 decimal value during the rendering of the Python bar charts.

## Key Design Decisions and Assumptions

In the design and implementation of PBD CA1 the following were the key design assumptions and decisions;

* Only one jar file is used for the Map-Reduce processing, and this is potentially able to manage multiple files across multiple languages. No hard coding of language names or named file subdirectories in HDFS is used in the Java code. The Java project would not need to be re-compiled to accommodate additional languages (limited to three for this assignment).
* Command line text and file name suffixes are used to alter the selected inputs to the Map-Reduce process and effect the content of the end result formats. All language file names must be in the format: <***bookname\_languagename***>, as seen in Section 2.3 of this report. For example, the file *IL\_Diavolo****\_Italian*** is one of the input books to the MR process in this assignment.
* Globs with a wildcard based on the filename language will pick up all language files for the MR process. In the assignment there are two book files per language but there is no limitation to adding more (and no need to amend source code).
* Two chained jobs are required. One to calculate the total occurrences of each character in the input text, and the second to use the output of the first job to calculate the total number of characters and hence the average letter frequencies (and output).
* The Python code is written in a Jupyter Notebook and also applies the same principle of having no hard coded references to languages. No file name conventions are required. The Python code reads any number of EXCEL files in the local directory of the notebook and calls a graph function. For simplicity, only three graphs are specifically rendered for this assignment, but the code could easily accommodate more language inputs.
* The output from Job2 is downloaded from the HDFS data store after each individual language MR process. The text file is transferred from the VM onto the host machine. The inbuilt EXCEL data ingestion wizard takes the text file and is used to create a three-column output of: ***language – character – frequency count***.
* No data manipulation takes place during the generation of the XL file, and no headers are added.
* The Python graph function takes two parameters; the dataframe with the language letter frequency information loaded from XL; and the name of the language, which is read directly from the dataframe.

# Map-Reduce: Java Code

## Driver

A description of the operations in this Java code class are given in Section 3.2.1.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## Job 1

### The Mapper

A description of the operations in this Java code class are given in Section 3.2.2.

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

### The Partitioner

A description of the operations in this Java code class are given in Section 3.2.3,

Graphical user interface, text, application, email

Description automatically generated

### The Reducer

A description of the operations in this Java code class are given in Section 3.2.4.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Job 2

### The Mapper

A description of the operations in this Java code class are given in Section 3.2.6.

Graphical user interface, text, application, email

Description automatically generated

### The Reducer

A description of the operations in this Java code class are given in Section 3.2.7,

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

# Python: Graph Analysis of MR Outputs

## Set Up Excel From HSFs Output

The final output from the Map-Reducer process is downloaded on the VM and then copied to the host machine.

The text file is loaded into Excel and the inbuilt wizard creates an \*.xlsx file output wit the data in three columns (no headers).

Graphical user interface, text, application

Description automatically generated

*Figure – Loading HDFS Output Into Excel*

All \*.xlsx files are then copied to the same local directory from which the Python Jupyter Notebook written for the assignment is stored.

## Python Jupyter Notebook

The Python code reads all locally stored \*.xlsx files (all of which are assumed to be in the HDFS output format) and loads this language data into separate Panda dataframes.

The notebook generates a character frequency graph for each language dataframe.

Below is the Jupyter Notebook code with generated output included;

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated with low confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, table

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

<image>

## Analysis Commentary

Commentary on each language graph is included in the output in Section 2.4 of the Python Jupyter Notebook.

A brief description is provided on how the distribution of characters for each language varies from English. The average letter frequency for English given in the brief for PBD Assignment 1 is used as the basis for this comparison.

Given that the assignment brief shows a letter frequency based on the Oxford English dictionary, it would be necessary to greatly increase the size of input for ‘foreign’ languages in our VM to give a more accurate comparison. Resource limitation for this CA meant that it was necessary to restrict the size of the Map-Reduce input.