Assignment 1

Name	Praveen Mohanprasad
Student Number	D18128998
Course	Data Science TU059
Year of programme	1st year
Module	Programming with Bigdata

Table of Contents

Configuration Setup and Assumptions

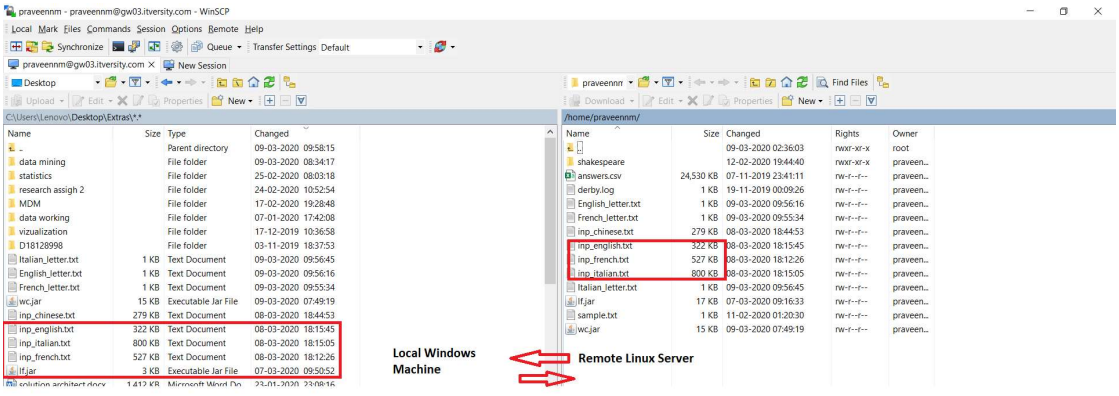
- Big data cluster, I use is a 5 node cluster plus a gateway node to submit jobs having a total memory capacity of 120 GB, with a total of 12 Vcores.
- We are using following 3 individual files (all less than 1 MB) from languages, English, French and Italian as inputs to compute letter frequencies,

  English - Decline of Science in England, **French** - Histoire des salons de Paris, **Italian** - Le rive della Bormida nel 1794
- In every language book, first few lines belong to English irrespective of the actual language of the book. But We consider that all the letters right from the first one in any language book belongs to that particular language and not English (first few lines), the letters in the first few lines are also counted as belonging to that particular language instead of English.
- Only one file per language is considered as input, So computation of average does not hold here.
- `mapreduce.output.textoutputformat.separator` is set as ";" in driver code to get output cols separated.

Loading Data into HDFS

- Getting the 3 files downloaded from the website, using **WINSCP** third party software, they are moved from local machine into the big data cluster gateway node.

Note : WinScp is a file transfer software which helps to transfer files between windows machine and linux servers, We need to login to the linux machine using the credentials, WinScp then provides a layout where we see our local windows machine directory on the left side and the linux machine directories on the right side, We can transfer files between these two just by dragging and dropping files between the layouts. A Screenshot of the software layout



- After logging into the gateway node of the bigdata cluster, we can see the files that were moved through winscp, resting inside a folder 'inp' our home directory (/home/praveennm).
- Having our files in the linux home directory of the gateway node, we need to push them into hdfs directory /user/praveennm/lf_inp, We use lf_inp as the hdfs directory containing all the three input files.

- So, we use Hadoop put command to push files from local linux server into hdfs,

  hdfs dfs -put /home/praveennm/inp/* /user/praveennm/lf_inp/

Running the job

Once we move the input files to the HDFS directory, after we are done with the map reduce program, we bundle the code files and dependencies into a executable jar, move it to the gateway node of the cluster, Trigger the job using the following command,

hadoop jar wc.jar com.dataflair.hd.wc.WordCount /user/praveennm/lf_inp/ /user/praveennm/lf_out /user/praveennm/temp

Arguments passed:

Argument 1 – HDFS input directory where the input files are placed (***/user/praveennm/lf_inp/***)

Argument 2 - HDFS directory where the output files are to be placed (***/user/praveennm/lf_out/***)

Argument 3 – HDFS directory to store the intermediate output (ie) the output of first mapreduce program (***/user/praveennm/temp***)

Job ran successfully as follows,

```
20/03/09 11:14:14 INFO mapreduce.Job:  map 0% reduce 0%
20/03/09 11:14:20 INFO mapreduce.Job:  map 50% reduce 0%
20/03/09 11:14:21 INFO mapreduce.Job:  map 100% reduce 0%
20/03/09 11:14:26 INFO mapreduce.Job:  map 100% reduce 100%
20/03/09 11:14:27 INFO mapreduce.Job: Job job_1565300265360_63148 completed succ
essfully
20/03/09 11:14:27 INFO mapreduce.Job: Counters: 50
```

Job Tracker UI for Successful completion of mapreduce job 1 :

Task Type	Total	Complete
Map	3	3
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	3
Reduces	0	0	1

Job Tracker UI for Successful completion of mapreduce job 2 :

Task Type	Total	Complete
Map	1	1
Reduce	3	3

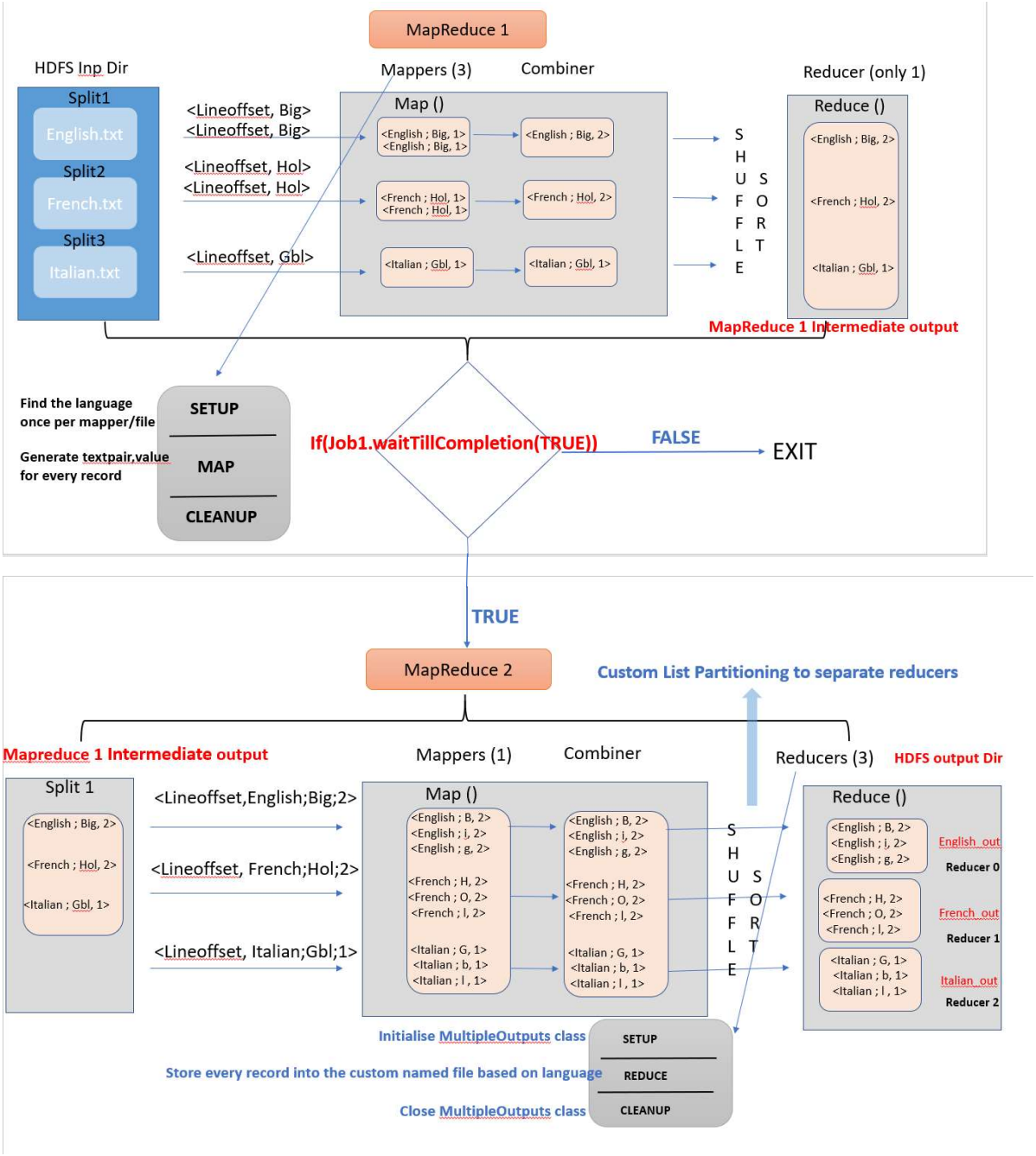
Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	3

Design Process

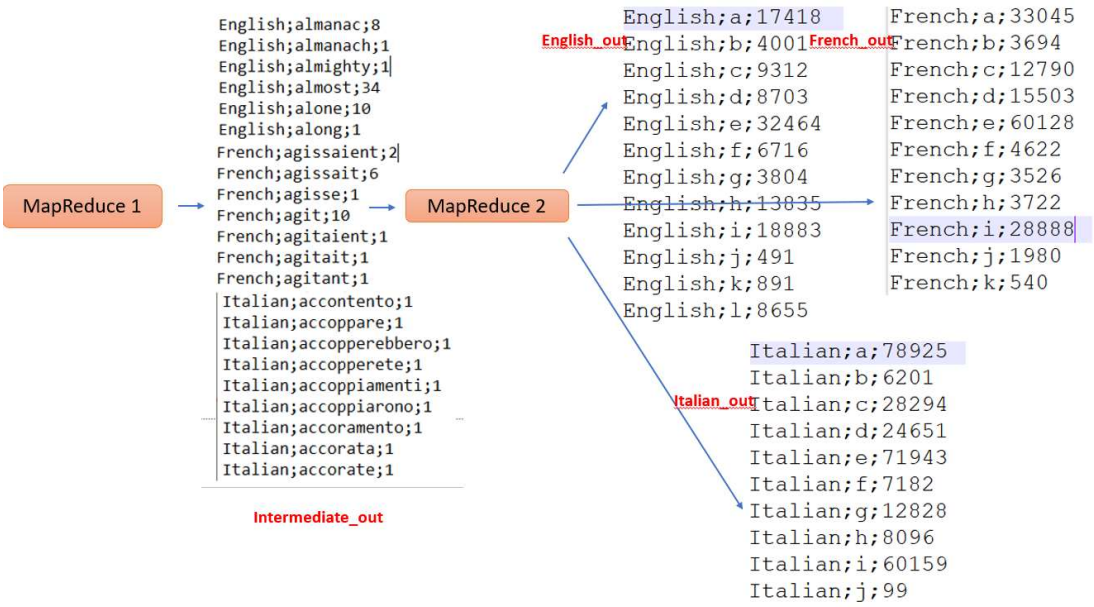
- Chained Mapreduce process is followed which chains two mapreduce jobs and the control flow is configured in such a way that the job 2 starts only when the job 1 is successful
- Mapreduce 1 is designed to find the frequency of words in every file with respect to the particular language.
- Mapreduce 2 is designed to take the output of mapreduce1 as the input and compute the frequency of letters in each word.

Process flow works as below where we can see the intermediate output of Reducer of Mapreduce 1 has the frequency of words which is taken as input by the Mapreduce 2 job and it computes the frequency of letters.

MapReduce Flow/Conceptual Diagram



Part of the outputs obtained,



Detailed Functionality

Mapreduce 1

is designed in such a way that we get intermediate output stored in HDFS which has three columns Language of the book, Word and the Frequency of the word.

Input – Output of **Mapreduce 1:**

Input – '/user/praveennm/lf_inp' – argument 1

Output – '/user/praveennm/temp' – argument 3

Mapper 1 includes the following,

1. A **Setup** method overriding the base class :: **A Dynamic Approach to find the language of the file**,

This is the method in mapper which gets triggered only once at the time of initialization, So this is used to find the language of the book and store it in a variable, *InputFileLanguage.* Every book has a key word "**Language:**" which tells the actual language of the book and it is present few lines away from the start. Every book is less than the split size(), So we get 1 mapper for every book file. When a split (one book file) enters the mapper, We get the name of the HDFS file using fileSplit API properties and read that file from our input HDFS directory, scan through it till we reach the word "Language:", save the value of it in the variable *InputFileLanguage.*

2. **Map** method returns key value as,

Key – TextPair – *InputFileLanguage;Word*

Value – IntWritable – 1.

3. **TextPair** – Since we need two parameters, language and word as key, A custom datatype has been created with proper getter setters having First parameter as language and second parameter as the actual word.

The Map method picks up every word from the line(Map method gets triggered for every new line), uses the variable, *InputFileLanguage* initialised in the setup method, uses the custom data type Textpair, returns language and word as the key and static number 1 as the value.

4. **Combiner ,** performs the same functionality as that of reducer 1 where it groups, aggregates the output of the map method in such a way that similar keys are grouped, performing the sum of all the static 1s of those similar keys. This process is done within that particular mapper's output in order to reduce the number of key-value pairs getting shuffled to the reducer over network for good performance.

5. **Partitioner** is left to the **default** Hash partitioner with number of reduce tasks as 1 (default value) as this is just an intermediate output of 1 file.

6. **Reducer ,** This receives the output of the combiner and pushes it to the reduce method. Here the same process as that of combiner is performed which is aggregation(sum) of values of similar keys but, with a difference that in reduce phase the grouping and aggregation of similar keys happen across all mappers. Output of reducer is stored in the intermediate HDFS directory (Third argument passed in the command) in the below format,

   Language;Word;Frequency

   Example: English;BigData;100

Mapreduce 2

It is designed in such a way that we get the final output stored in the output HDFS directory which has three columns, Language;Letter;Frequency.

Input – Output of **Mapreduce 2:**

Input – '/user/praveennm/temp' (Output directory of Mapreduce1) – argument 3

Output – '/user/praveennm/lf_out' – argument 2

1. **Mapper** , Records are read line by line, from the above input directory, map method is triggered and returns key value as,

Key – TextPair – *InputFileLanguage;*Each letter of the word

Value – IntWritable – Frequency of the word (Same column is passed without any change).

Example: Input record -> **English;Big;100**

Output records -> **English;B;100**

English;i;100

English;g;100

2. **Combiner** , It performs same as that of combiner 1 and simply copies the functionality of reducer .

3. **Partitioner** is custom defined as List Partitioner with **Number of reduce tasks** set as 3.

The Motive is to get separate files for every language. Since we pass three language files as input we set number of reducers as 3 so that we get three out files. Partitioner logic is given in such a way that every key value goes to a particular reducer based upon the language key as follows,

English;B;100 ---> Reducer 0

French;A;200 ---> Reducer 1

Italian;F;200 ---> Reducer 2

4. **Reducer** , This receives the output of the combiner 2 and pushes it to the reduce method. Here the same process as that of combiner is performed which is aggregation(sum) of values of similar keys but, with a difference that in reduce phase the grouping and aggregation of similar keys happen across all mappers.

Reducer includes the following,

5. A **Setup** method where we initialise **MultipleOutputs** class, This is used mainly to derive multiple outputs based on the language, our custom partitioner already does the job of pushing language specific records to the individual reducer and this multiple outputs class is used following the reducer's output giving the language name automatically for the output file based on the language content inside it.

Example – French_out* for French language letter file.

Initially we had no multiple outputs class but later we added it which rendered better usability,

This makes it easy for us to identify the output file as follows,

```
-rw-r--r--   2 praveennm hdfs        393 2020-03-09 03:47 /user/praveennm/lf_out55/part-r-00000
-rw-r--r--   2 praveennm hdfs        375 2020-03-09 03:47 /user/praveennm/lf_out55/part-r-00001
-rw-r--r--   2 praveennm hdfs        399 2020-03-09 03:47 /user/praveennm/lf_out55/part-r-00002
-rw-r--r--   2 praveennm hdfs        886 2020-03-09 03:47 /user/praveennm/lf_out55/part-r-00003
```

```
[praveennm@gw03 ~]$ hadoofs -ls /user/praveennm/lf_out56
Found 9 items
-rw-r--r--   2 praveennm hdfs        886 2020-03-09 03:50 /user/praveennm/lf_out56/ChineseCharactersetencoding_out_-r-00003
-rw-r--r--   2 praveennm hdfs        393 2020-03-09 03:50 /user/praveennm/lf_out56/English_out_-r-00000
-rw-r--r--   2 praveennm hdfs        375 2020-03-09 03:50 /user/praveennm/lf_out56/French_out_-r-00001
-rw-r--r--   2 praveennm hdfs        399 2020-03-09 03:50 /user/praveennm/lf_out56/Italian_out_-r-00002
```

Output of reducer is stored in the final output HDFS directory (second argument passed in the hadooop command) in the below format,

English_out.txt	French_out.txt	Italian_out.txt
Language;Letter;Frequency	**Language;Letter;Frequency**	**Language;Letter;Frequency**
English;B;1000	French;A;1500	Italian;B;2000
English;G;2000	French;V;2000	Italian;H;800

6. A **Cleanup** method to close the multipleoutputs class.

7. **Counters:**

The requirement is to find the frequency of letters in the books but the books also contain numbers in them, So the occurrences of digits are not included and counted in the program. But, to keep track of the numbers, A counter (`numbers.DIGITS_OCCURANCE`) has been defined which gets incremented whenever a digit occurs in the file. As a result there were **1,401** occurrences of digits in all the language files.

Presence of the designed counter in the job tracker's web UI,

		Name	Map	Reduce	Total
com.dataflair.hd.wc.LetterReducer$numbers	DIGITS_OCCURANCE		0	1,401	1,401
		Name	Map	Reduce	Total
File Input Format Counters	Bytes Read		564,674	0	564,674
		Name	Map	Reduce	Total
File Output Format Counters	Bytes Written		0	2,053	2,053

Highlight of the Advanced features and functionalities implemented

A **Setup & Cleanup method** in mapreduce1 mapper – Logic implemented here **Dynamically** finds the language of the book without hardcoding it, This becomes so dynamic and advantageous as in future we can add n number of files of the same format (from http://www.gutenberg.org/) in the input directory, our mapreduce will dynamically find the language of the book and process it to compute letter frequency.

Counters – Gets the occurrences of the digits.

Combiner – To reduce the transfer of data over network by grouping and aggregating within mapper.

Partitioner – Custom List Partitioner – To move language specific files to individual reducers and get individual language specific outputs

MultipleOutputs – To enhance the usability, we use this to get individual language specific outputs with **file names prefixed automatically with the language of it,** In future if we add a Chinese book we automatically get the output with file name prefixed as **Chinese***

Cutom Datatypes – TextPair – To frame keys with not only one column but with more than one.

Design decisions and Justifications after alternatives

- Choosing Chained map reduce involving series of 2 mapreduce programs, one for word count, one for letter count, instead of one mapreduce directly computing letter count – This increased the performance, as by the chained process, we follow a hierarchical flow of count (ie) First words then letters.
- Using List Partitioner over other types – This type of partitioner is suitable here as we have a category of languages having three values (English, French and Italian).
- Using TextPair instead of Text type – Initially we tried with text type concatenating language and the word with a separator, but this was difficult as we had to split the text each and every time while processing and it is not readable, So adopted Textpair type.

- Using reducer logic for the combiner – This reduced the data to be shuffled over network and since it is just the sum of counts, we used the reducer logic itself here.
- Using MultipleOutputClass - This made it possible to give right names to the output files automatically before which we had default names (part-r-00000), which made our partitioner logic of getting language specific records in separate files useless. After using multipleOutputClass, Partitioner logic is complemented.
- Adding the language fetch logic in mapper's setup method instead of putting inside map method – First when we had this logic inside map, for every record it was deriving the language which is unwanted. Considering the setup method's functionality an dadding language logic inside setup made the language fetch only once per mapper which improved the performance as the language fetch logic scans a hdfs file which is a read intensive process.

Analysis of Output files using R

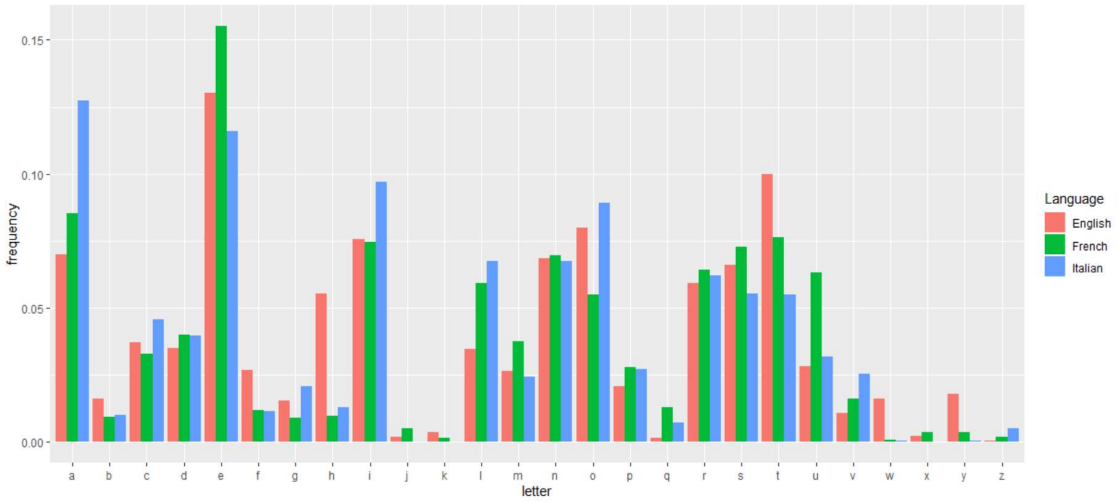
Using HDFS get command, the output files are moved to local linux server and then using WinSCP, these files are moved to our windows machine,

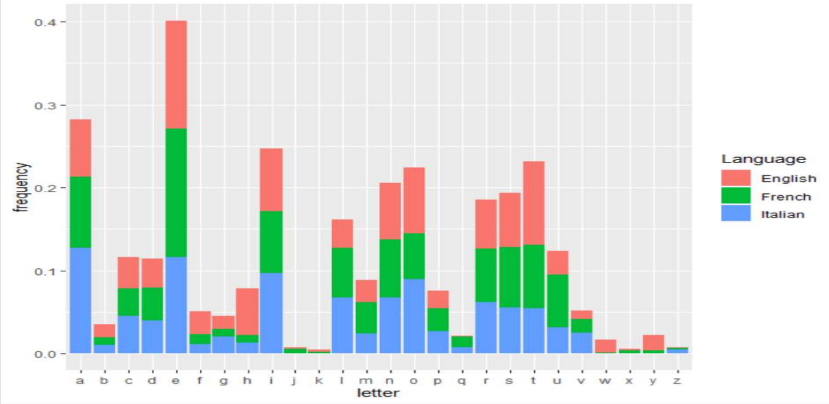
Hdfs dfs -get /user/praveennm/lf_out/* /home/praveennm/

These files are then converted to data frames, merged together into a single dataframe. Grouped (position = dodge) and stacked Bar chart visualising the comparison of frequency of letters in every language book,

we used R library ggplot to build the chart using the merged dataframe.

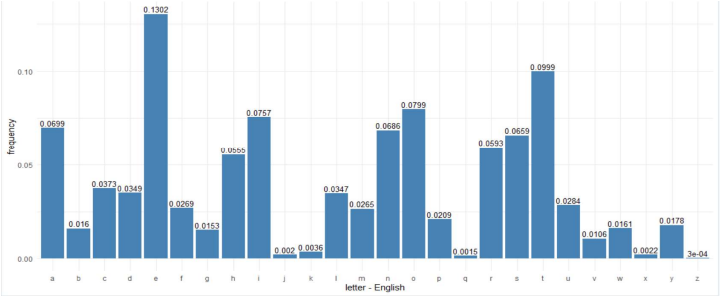
Insight - Letter 'e' has been occurring the most in all the selected language books and letter 'z' has been occurring least number of times followed by 'j k x' in all languages. Letter 'a' and 'i' have the highest frequencies next to 'e' for all the languages.



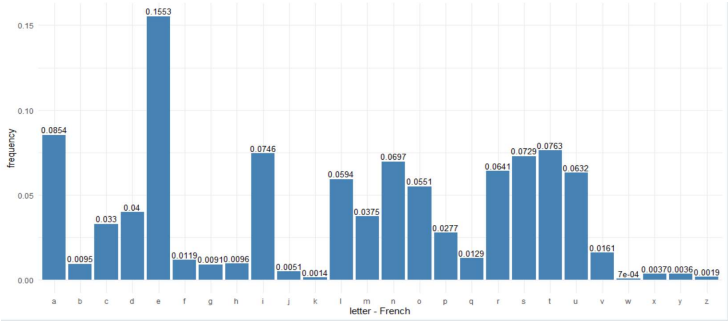


1. Following charts visualise the frequency of letters individually in separate charts for each language just to know the actual numbers for each letter

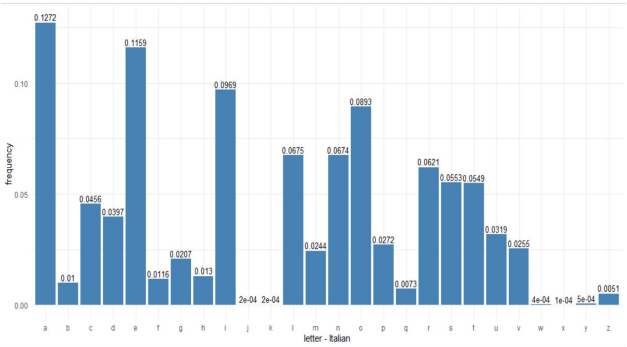
English : A ggplot Bar chart where ‘e’ has the highest frequency and ‘z’ the lowest



French : A ggplot Bar chart where ‘e’ has the highest frequency and ‘w’ the lowest



Italian : A ggplot Bar chart where ‘a’ has the highest frequency and ‘j k’ the lowest



Note: Only important facets of every segment of the map reduce code is attached below, Runnable code is attached in separate files and also as a executable jar in the attachment.

Code Snippets

Driver

```
public class WordCount {
    public static void main(String[] arg0) throws Exception { /*Initialising config for first job*/
        Configuration conf = new Configuration();
        conf.set("mapreduce.output.textoutputformat.separator", ";"); // Setting the seperator of the output file as ';'
        String[] otherArgs = new GenericOptionsParser(conf, arg0).getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("Usage: wordcount <in> [<in>...] <out>");
            System.exit(2);
        }
        /* Job 1 configuration */
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(combiner.class);
        job.setReducerClass(IntSumReducer.class);
        job.setMapOutputKeyClass(TextPair.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        /* input output configuration of mapreduce 1 */
        conf.set("inp_path", arg0[0]);
        FileInputFormat.addInputPath(job, new Path(arg0[0]));
        FileOutputFormat.setOutputPath(job, new Path(arg0[2]));
        /*Initialising config for second job*/
        Configuration conf2 = new Configuration();
        conf2.set("mapreduce.output.textoutputformat.separator", ";");
        /* Job 2 configuration */
        Job job2 = Job.getInstance(conf2);
        job2.setJarByClass(WordCount.class);
        job2.setMapperClass(LetterMapper.class);
        job2.setMapOutputKeyClass(TextPair.class);
        job2.setMapOutputValueClass(IntWritable.class);
        job2.setCombinerClass(combiner2.class);
        job2.setPartitionerClass(ListPartitioner.class); // Cutom partitioner
        job2.setNumReduceTasks(3); // Setting up 3 reducers for 3 files
        job2.setReducerClass(LetterReducer.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(IntWritable.class);
        /* input output configuration of mapreduce 2 */
        FileInputFormat.addInputPath(job2, new Path(arg0[2]));
        FileOutputFormat.setOutputPath(job2, new Path(arg0[1]));
       // Check if job 1 is successful then proceed to job 2 :: Chaining Mapreduce
        if (job.waitForCompletion(true)) {
            ControlledJob cJob2 = new ControlledJob(conf2);
            cJob2.setJob(job2);
            // Adding job 2 to the control flow
            JobControl jobctrl = new JobControl("jobctrl");
            jobctrl.addJob(cJob2);
            cJob2.addDependingJob(cJob1);
            jobctrl.run();
            while (!jobctrl.allFinished()) {
                System.out.println("Still running...");
                Thread.sleep(5000);
            }
            System.out.println("done");
            jobctrl.stop();
            }
```

Mapper :: Mapreduce1

```
public class TokenizerMapper extends Mapper<LongWritable, Text, TextPair, IntWritable> {

    /* Initialising textpair */
    private static TextPair mapOutputKey = new TextPair();
    String inputFileLanguage;

    /*USing setup method to find the language of the file*/
    @Override
    protected void setup(Context context) throws IOException, InterruptedException
    {
        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        // getting the split file name using fileSplit API //
        inputFileLanguage = fileSplit.getPath().getName();
        Configuration conf = context.getConfiguration();

        /*Setting up the HDFS Config files to open HDFS*/
        conf.addResource(new Path("/etc/hadoop/conf/core-site.xml"));
        conf.addResource(new Path("/etc/hadoop/conf/hdfs-site.xml"));

        /* Opening HDFS and scanning the HDFS File*/
        FileSystem fs = FileSystem.get(conf);
        FileStatus[] status = fs.listStatus(new Path("hdfs://nn01.itversity.com:8020/user/praveennm/lf_inp"));

        /*Iterating through the files in the input hdfs directory to find the actual split file */
        for (int i = 0; i < status.length; i++) {
            FSDataInputStream inputStream = fs.open(status[i].getPath());
            String content = IOUtils.toString(inputStream, "UTF-8");

            /* check if the split file and the file in the hdfs input dir matches, then find the language */
            if(inputFileLanguage.equals(status[i].getPath().toString().split("/")[status[i].getPath().toString().split("/").length-1]))
            inputFileLanguage = content.substring(content.indexOf("Language:"),content.indexOf("****")).replaceAll(" ","").replaceAll("\\n","").replaceAll("\\r","").split(":")[1];

        }
    }

    public void map(LongWritable lineOffset, Text record, Context context) throws IOException, InterruptedException {
        String s = record.toString().toLowerCase();
        /* getting every word in the split */
        for (String word : s.split("\\W+")) {
        mapOutputKey.setFirst(inputFileLanguage);
        mapOutputKey.setSecond(word);

        /* sample output - English Big 100 */
       context.write(mapOutputKey, new IntWritable(1));

    }
    }
```

Reducer :: Mapreduce 1

```
public class IntSumReducer extends Reducer <TextPair, IntWritable, Text, IntWritable>
{
    private IntWritable result = new IntWritable();
    /* Initialising seconf text Pair */
    private static TextPair mapOutputKey1 = new TextPair();

    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
    {   /*Aggregation (sum) of counts of similar keys */
        int sum = 0;
        for (IntWritable val : values)
        {
            sum += val.get();
        }
        result.set(sum);
        context.write(new Text(key.getFirst()+";"+key.getSecond()), result);
    }
```

Mapper :: Mapreduce 2

```
public class LetterMapper extends Mapper<LongWritable, Text, TextPair, IntWritable> {
    /* Initialising textpair */
    private static TextPair mapOutputKey = new TextPair();
    String inputFileLanguage;
    String cacheContent;
    public void map(LongWritable lineOffset, Text record, Context context) throws IOException, InterruptedException {

        System.out.println("cache:" + cacheContent);

        // Getting the Word part alone
        String word = record.toString().split(";")[1];

        //Iterating through the length of the word to get each character of the word and pass it to reducer
        for (int i = 0; i < word.length(); i++) {
            mapOutputKey.setFirst(record.toString().split(";")[0]); // Language
            mapOutputKey.setSecond(String.valueOf(word.charAt(i)));// Getting the letter of each word
            context.write(mapOutputKey, new IntWritable(Integer.valueOf(record.toString().split(";")[2])));// English A 100 (sample)

        }


    }
```

Reducer :: Counter :: Mapreduce 2

```
public class LetterReducer extends Reducer <TextPair, IntWritable, Text, IntWritable>
{   // Counter for Digits Occurence
    public enum numbers {
        DIGITS_OCCURANCE
        }

    private IntWritable result = new IntWritable();
    /* Initialising textpair */
    private static TextPair mapOutputKey1 = new TextPair();
    /* Initialising multipleOutputs inside setup */
    private MultipleOutputs<Text, IntWritable> multipleOutputs;

    protected void setup(Context context) throws IOException, InterruptedException {
        multipleOutputs = new MultipleOutputs<Text, IntWritable>(context);
    }

    public void reduce(TextPair key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
    {
        int count_ = 0;
        char c = key.getSecond().toString().toLowerCase().charAt(0);
        char d = key.getSecond().toString().charAt(0);
        String regex = "[0-9]+";
        String NotNumber = "^[0-9]+";
        for (IntWritable record : values)
        {
            // Incrementing counters for digit occurrence
            if(key.getSecond().toString().matches(regex))
            context.getCounter(numbers.DIGITS_OCCURANCE).increment(1);
            count_ = count_+record.get();

        }
        if((c >= 'a' && c <= 'z'))
        { // Giving the language name as the prefix to the output file using multipleoutputs
        String basePath = key.getFirst().toString()+"_out_";
        multipleOutputs.write(new Text(key.getFirst().toString()+";"+key.getSecond().toString()), new IntWritable(count_), basePath);
        }
        }


    protected void cleanup(Context context) throws IOException, InterruptedException {
        //Terminating multipleOutouts
        multipleOutputs.close();
}
}
```

Custom Partitioner :: Mapreduce 2

```
public class ListPartitioner extends Partitioner<TextPair, IntWritable>{
    //Partitioning logic to push each language letter to different reducer
    @Override
    public int getPartition(TextPair arg0, IntWritable arg1, int arg2) {
        if(arg0.getFirst().toString().toLowerCase().contains("english"))
        return 0;
        if(arg0.getFirst().toString().toLowerCase().contains("french"))
        return 1;
        if(arg0.getFirst().toString().toLowerCase().contains("italian"))
        return 2;
        return 3;

    }
```

Analysis R code

```
library(ggplot2)
#Read English file into Dataframe
English <- read.delim("English_letter.txt",header = FALSE,sep=";")
#Adding header
colnames(English)<-c('Language','letter','counts')

#calculating Frequency column
English<-
  mutate(English,frequency=round(counts/sum(counts),4))
#Bar chart for english letters
ggplot(data=English, aes(x=letter, y=frequency)) +
  geom_bar(stat="identity", fill="steelblue")+
  xlab("letter - English")+
  geom_text(aes(label=frequency), vjust=-0.3, size=3.5)+
  theme_minimal()

#Read French file into Dataframe
French <- read.delim("French_letter.txt",header = FALSE,sep=";")
#Adding header
colnames(French)<-c('Language','letter','counts')
#calculating Frequency column
French<-
  mutate(French,frequency=round(counts/sum(counts),4))
#Bar chart for French letters
ggplot(data=French, aes(x=letter, y=frequency)) +
  geom_bar(stat="identity", fill="steelblue")+
  xlab("letter - French")+
  geom_text(aes(label=frequency), vjust=-0.3, size=3.5)+
  theme_minimal()

#Read Italian file into Dataframe
Italian <- read.delim("Italian_letter.txt",header = FALSE,sep=";")
#Adding header
colnames(Italian)<-c('Language','letter','counts')
#calculating Frequency column
Italian<-
  mutate(Italian,frequency=round(counts/sum(counts),4))
#Bar chart for Italian letters
ggplot(data=Italian, aes(x=letter, y=frequency)) +
  geom_bar(stat="identity", fill="steelblue")+
  xlab("letter - Italian")+
  geom_text(aes(label=frequency), vjust=-0.3, size=3.5)+
  theme_minimal()

#Merge the above three dataframes to get all the languages in a single dataframe
All_languages <- merge(English,French,all = TRUE)
All_languages <- merge(All_languages,Italian,all=TRUE)

#Stacked Bar chart to compare frequency of letters across all the three languages
ggplot(All_languages, aes(fill=Language, y=frequency, x=letter)) +
  geom_bar(position="dodge", stat="identity")

#Grouped Bar chart to compare frequency of letters across all the three languages
ggplot(All_languages, aes(fill=Language, y=frequency, x=letter)) +
  geom_bar(position="dodge", stat="identity")
```