Introduction:

The aim of this part of the project is to perform a Hadoop Inverted Index MapReduce operation of an input corpus consisting of multiple files having huge amounts of textual data(novels/books). The output generated would have a word as the key and list of files in which the word is present as the value.

With the use of Hadoop, each text file can be processed concurrently across multiple nodes, significantly reducing processing time compared to sequential processing. Because of its scalability, adding more nodes to the cluster is an easy way to handle increasing data volumes. In the event of a node loss, Hadoop's fault tolerance reassigns workloads to ensure data reliability, and its optimized storage, controlled by the Hadoop Distributed File System (HDFS), facilitates effective data accessibility. By breaking down the difficult task into smaller, more manageable pieces, the MapReduce paradigm improves efficiency and performance. In the end, this method uses big data analytics to produce an indexed structure that facilitates rapid searches and in-depth analytical insights about word frequencies and distributions across novels.

Dataset Selection:

I have chosen the **data** folder in the given **data.rar** file as the input corpus. It consists of 18 text files. The text files are books or novels from famous authors. These files have various sizes ranging from a few KB up to 4 MB. hadoop has the functionality of parallel processing multiple files and producing outputs for large amounts of data. Since these books have a large amount of words in them, they are an ideal fit to perform inverted index operation on.

Code Implementation:

Code Location: finalProject\src\main\java\org\example
Jar Location: finalProject\out\artifacts\finalProject_jar

The way I have designed the Inverted Index mapreduce function is as follows:

1. Map function
    a. Mapper reads the input files from the `data` folder sequentially after splitting the files
    b. Within each file it reads the text line by line
    c. Since we are performing an inverted index operation on input files consisting of a variety of words with several punctuation marks, we need to first preprocess the data.
    d. This is done by normalizing each line as an to lowercase(“the” and “The” should be considered same), removing punctuation marks and finally tokenizing each line
    e. I have also considered one more case where we might be having some numerical values as well such as page numbers as a unique key. Since that won’t be very insightful I decided to skip the words that were encountered as numeric values using regex and only added alphabetical words with their corresponding filenames as output of the mapper(key value pairs)
    f. Code:

```
public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException{

   FileSplit currentSplit = ((FileSplit) context.getInputSplit());
   String filenameStr = currentSplit.getPath().getName();
   filename = new Text(filenameStr);

   String line = value.toString();

   // case normalization
   String normalizedLine = Normalizer.normalize(line,
Normalizer.Form.NFD).toLowerCase();

   // remove punctuation
   String newLine = normalizedLine.replaceAll("[\\p{Punct}]", "");

   // line tokenization
   StringTokenizer tokenizer = new StringTokenizer(newLine);

   while (tokenizer.hasMoreTokens()) {
       String token = tokenizer.nextToken();

       // Check if the token contains at least one alphabetical character
       if (token.matches(".*[a-zA-Z].*")) {
           // Ensure the token is composed solely of letters
           if (token.matches("[a-zA-Z]+")) {
               word.set(token);
               context.write(word, filename);
           }
       }
   }

}
```

2. Reduce function
    a. In the reducer we aim to consolidate/aggregate the filenames(values) to their corresponding words(keys)
    b. The mapper generates an output such that the filename is repeated against a word as many times as the word has occurred in that file and we want to avoid this repetition
    c. We have modified the logic for the reducer such that the filenames are stored in a hashset and are finally appended next to the word
    d. The output of the reducer is: word | filename1.txt | filename2.txt | …

```
public void reduce(final Text key, final Iterable<Text> values,
                   final     Context     context)     throws     IOException,
InterruptedException {

   Set<String> uniqueFiles = new HashSet<>(); // To store unique file names
for each key

   StringBuilder stringBuilder = new StringBuilder();
```

```
    for (Text value : values) {
        String filename = value.toString();

        // Modify code to append a filename after the key only once.
        // If the word is encountered again, the filename won't be added to
the value list again.
        // Check if the file name is already seen for this key
        if (!uniqueFiles.contains(filename)) {
            stringBuilder.append(filename).append(" | ");
            uniqueFiles.add(filename); // Add filename to the set to mark it
as seen
        }
    }

    if (stringBuilder.length() > 3) {
        // Remove the last " | " from the StringBuilder
        stringBuilder.setLength(stringBuilder.length() - 3);
    }

    context.write(key, new Text(stringBuilder.toString()));
}
```

3. Mapreduce function
    a. In the main method, we instantiate a Job using the `Job.getInstance()` method and provide it a configuration and a job name
    b. This function forms the integration of mapper and reducer and performs their required operations one after another
    c. It is here that we define how we would be passing the input and output paths

```
public int run(String[] args) throws Exception{

    if(args.length !=2){
        System.err.println("Invalid Command");
        System.err.println("Usage: <input path> <output path>");
        return -1;
    }

    Job job = Job.getInstance(getConf());
    job.setJobName("invertedIndex");
    job.setJarByClass(InvertedIndex.class);


job.getConfiguration().set("mapreduce.output.textoutputformat.separator", "
| ");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

       // defining input and output files
```

```
    Path inputFilePath = new Path(args[0]);
    Path outputFilePath = new Path(args[1]);
    FileInputFormat.addInputPath(job, inputFilePath);
    FileOutputFormat.setOutputPath(job, outputFilePath);

    FileInputFormat.setInputDirRecursive(job, true);

    return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new InvertedIndex(), args);
    System.exit(exitCode);
}
```

Implementation:

Creating parent directory in / folder

```
anujamerwade@Anuja:/$ hdfs dfs -mkdir final_project
anujamerwade@Anuja:/$ hdfs dfs -ls
Found 3 items
drwxrwxrwx   - anujamerwade supergroup          0 2023-10-07 16:59 assignment_data
drwxr-xr-x   - anujamerwade supergroup          0 2023-10-22 02:21 asst
drwxr-xr-x   - anujamerwade supergroup          0 2023-12-08 12:21 final_project
```

Copying data folder from local to hdfs

```
anujamerwade@Anuja:/$ hadoop fs -copyFromLocal /mnt/c/Users/anuja/Downloads/Anuja/Sem_3/BDA/FinalProject/data
final_project/data
anujamerwade@Anuja:/$ hdfs dfs -ls final_project
Found 3 items
drwxr-xr-x   - anujamerwade supergroup          0 2023-12-08 12:45 final_project/data
```

MapReduce program executed successfully

```
anujamerwade@Anuja:/$ hadoop jar /mnt/c/Users/anuja/Downloads/Anuja/Sem_3/BDA/Asst4/finalProject/out/artifacts/finalProject_jar/finalProject.jar
 final_project/data final_project/output/finalop
2023-12-08 17:06:31,774 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-12-08 17:06:32,318 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/anujamerwade/.staging/jo
b_1702052853215_0011
2023-12-08 17:06:37,193 INFO input.FileInputFormat: Total input files to process : 18
2023-12-08 17:06:37,737 INFO mapreduce.JobSubmitter: number of splits:18
2023-12-08 17:06:38,487 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1702052853215_0011
2023-12-08 17:06:38,487 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-12-08 17:06:38,794 INFO conf.Configuration: resource-types.xml not found
2023-12-08 17:06:38,794 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-12-08 17:06:39,145 INFO impl.YarnClientImpl: Submitted application application_1702052853215_0011
2023-12-08 17:06:39,205 INFO mapreduce.Job: The url to track the job: http://Anuja.:8088/proxy/application_1702052853215_0011/
2023-12-08 17:06:39,206 INFO mapreduce.Job: Running job: job_1702052853215_0011
2023-12-08 17:06:50,439 INFO mapreduce.Job: Job job_1702052853215_0011 running in uber mode : false
2023-12-08 17:06:50,442 INFO mapreduce.Job:  map 0% reduce 0%
2023-12-08 17:07:12,010 INFO mapreduce.Job:  map 6% reduce 0%
2023-12-08 17:07:13,171 INFO mapreduce.Job:  map 19% reduce 0%
2023-12-08 17:07:14,182 INFO mapreduce.Job:  map 28% reduce 0%
2023-12-08 17:07:15,195 INFO mapreduce.Job:  map 33% reduce 0%
2023-12-08 17:07:23,315 INFO mapreduce.Job:  map 39% reduce 0%
2023-12-08 17:07:24,334 INFO mapreduce.Job:  map 50% reduce 0%
2023-12-08 17:07:25,341 INFO mapreduce.Job:  map 61% reduce 0%
2023-12-08 17:07:32,599 INFO mapreduce.Job:  map 72% reduce 0%
2023-12-08 17:07:33,617 INFO mapreduce.Job:  map 83% reduce 0%
2023-12-08 17:07:34,625 INFO mapreduce.Job:  map 89% reduce 30%
2023-12-08 17:07:37,656 INFO mapreduce.Job:  map 100% reduce 30%
2023-12-08 17:07:38,667 INFO mapreduce.Job:  map 100% reduce 100%
2023-12-08 17:07:40,704 INFO mapreduce.Job: Job job_1702052853215_0011 completed successfully
2023-12-08 17:07:41,062 INFO mapreduce.Job: Counters: 55
        File System Counters
                FILE: Number of bytes read=4026096
```

Output file in HDFS

```
anujamerwade@Anuja:/$ hadoop fs -ls final_project/output/finalop
Found 2 items
-rw-r--r--   1 anujamerwade supergroup          0 2023-12-08 17:07 final_project/output/finalop/_SUCCESS
-rw-r--r--   1 anujamerwade supergroup    3410151 2023-12-08 17:07 final_project/output/finalop/part-r-00000
```

Output file copied to local

```
anujamerwade@Anuja:/$ hadoop fs -copyToLocal final_project/output/finalop/part-r-00000 /mnt/c/Users/anuja/Down
loads/Anuja/Sem_3/BDA/Asst4/finalProject/invertedindex_output
anujamerwade@Anuja:/$
```

Results:

Stored the final output in **invertedindex_output** file in local machine(provided in submission)

Sample output:

a | blake-poems.txt | burgess-busterbrown.txt | shakespeare-caesar.txt | austen-persuasion.txt | edgeworth-parents.txt | whitman-leaves.txt | austen-sense.txt | melville-moby_dick.txt | bryant-stories.txt | chesterton-thursday.txt | austen-emma.txt | carroll-alice.txt | shakespeare-macbeth.txt | chesterton-brown.txt | chesterton-ball.txt | shakespeare-hamlet.txt | milton-paradise.txt | bible-kjv.txt
aaron | bible-kjv.txt | milton-paradise.txt
aaronites | bible-kjv.txt
aarons | bible-kjv.txt | milton-paradise.txt
ab | edgeworth-parents.txt
aback | melville-moby_dick.txt | chesterton-ball.txt
abaddon | bible-kjv.txt
abaft | melville-moby_dick.txt
abagtha | bible-kjv.txt
abana | bible-kjv.txt
abandon | whitman-leaves.txt | milton-paradise.txt | melville-moby_dick.txt
abandond | whitman-leaves.txt
abandoned | melville-moby_dick.txt | edgeworth-parents.txt | chesterton-thursday.txt | austen-sense.txt | milton-paradise.txt | chesterton-ball.txt | chesterton-brown.txt
abandonedly | melville-moby_dick.txt
abandoning | whitman-leaves.txt
abandonment | whitman-leaves.txt | melville-moby_dick.txt | chesterton-thursday.txt
abarim | bible-kjv.txt | milton-paradise.txt
abase | whitman-leaves.txt | bible-kjv.txt
abased | bible-kjv.txt | melville-moby_dick.txt | whitman-leaves.txt
abasement | melville-moby_dick.txt
abashd | whitman-leaves.txt
abashed | milton-paradise.txt | edgeworth-parents.txt | melville-moby_dick.txt
abasing | bible-kjv.txt
abassin | milton-paradise.txt
abate | melville-moby_dick.txt
abated | melville-moby_dick.txt | milton-paradise.txt | bible-kjv.txt | edgeworth-parents.txt
abatement | melville-moby_dick.txt | austen-sense.txt
abating | melville-moby_dick.txt

abba | bible-kjv.txt
abbana | milton-paradise.txt
abbey | edgeworth-parents.txt | austen-emma.txt | chesterton-ball.txt
abbeyland | austen-sense.txt
abbeymill | austen-emma.txt
abbeyoh | austen-emma.txt
abbots | austen-emma.txt
abbreviate | melville-moby_dick.txt

Conclusion:

Creating an inverted index from a library of novels using Hadoop's MapReduce framework provides a reliable and scalable method for examining large amounts of textual material. This procedure effectively processes many text files in parallel by utilizing Hadoop's distributed computing capability, producing an index that associates words with the documents in which they appear. With the use of this index, one may conduct rapid searches and comprehensive textual analysis of all the novels, gaining important insights on word distributions, frequencies, and document associations. Moreover, Hadoop's fault tolerance, scalability, and optimized storage capabilities guarantee dependable and effective management of massive text data, enabling scholars, analysts, and applications to effortlessly extract significant insights and patterns from vast literary collections. All things considered, the use of Hadoop's MapReduce to create an inverted index on a novel repository is a strong and scalable method to enable thorough text analysis and information retrieval in the field of big data processing.

However there could be some limitations such as:

1. If the volume of data increases the resource requirements would increase substantially resulting in higher costs
2. Handling unstructured data could get challenging in terms of data preprocessing techniques like tokenization or multi-language encoding
3. As the number of documents grows, the inverted index's size increases, potentially leading to longer search times and increased storage requirements. Maintenance and updating of the index could become resource-intensive.

Future scope:

1. Implement advanced text preprocessing techniques to improve the quality of the inverted index, such as stemming, lemmatization, entity recognition, or handling multi-language content.
2. Integrate the inverted index with advanced analytical tools or natural language processing (NLP) techniques to extract deeper insights, perform sentiment analysis, or create recommendation systems based on the content.
3. Explore methodologies to process text data in real-time or streaming environments, enabling continuous updates to the inverted index for dynamic content.

References:

1. Ajit Kumar Mahapatra, Sitanath Biswas, Inverted indexes: Types and techniques, *July 2011 International Journal of Computer Science Issues 8(4)*

2. Giulio Ermanno Pibiri, Rossano Venturini, Techniques for Inverted Index Compression, *ACM Computing SurveysVolume 53 Issue 6 Article No.: 125 pp 1–36 https://doi.org/10.1145/3415148*
3. Zhi-Hong Deng, Beyond the Inverted Index, *DOI:10.13140/RG.2.2.11993.26726*