Project Report: Inverted Index using Hadoop MapReduce

Objective: To implement a Hadoop MapReduce job that takes the input corpus and generates an inverted index. The output will be a set of key-value pairs, where the key is a term (word), and the value is a list of document identifiers where the term appears.

Introduction

- **Aim**
  - Inverted Index is a HashMap like data structure that stores mapping between a word and the set of documents where this word is located. It is mainly used in document retrieval systems such as search engines. The main aim of my project is to demonstrate the creation of inverted index using Hadoop's MapReduce programming model.
- **Objective**
  - To process given set of text documents using MapReduce model and generating an inverted index as the outcome.
  - To explore and analyze the performance and other different aspects of Hadoop in processing big data.
  - To explore different ways to store and retrieve data in HDFS
- **Overview of Research topic**
  - **Research topic I selected : optimize data storage and retrieval performance in HDFS**
  - I worked as a Database Administrator for 2 years. I am curious in Data Science, and I understand the importance of efficient storage methods for faster retrieval. Hence, I chose this research topic.
  - Hadoop is responsible for handling large datasets across distributed environments. While MapReduce is responsible for data processing on large clusters providing efficient and parallel operations. The large dataset is split into many blocks and stored across different nodes in Hadoop. In this digital age, there is a lot of data generated each day through different watches, mobiles, sensors etc. However, with the growing data over years, there is need for optimized methods for storing the large datasets and retrieving it faster and efficiently to match the speed of new age
- **Significance of using Hadoop for big data processing and analytics**
  - **Open source:** Hadoop is open-source framework giving the freedom to users to modify the code according to their needs as each company as different requirements.
  - **Scalability:** It offers scalability as we can increase the nodes to process large amounts of data
  - **High availability:** If any of the one node goes down, data can be retrieved from another node.
  - **Flexibility:** It can deal with any kind of dataset - structed(relational databases), unstructured(images, videos, streaming data), semi structured(xml, Json)
  - **Distributed File system and parallel processing:** Hadoop stores a large file in smaller blocks which are stored in different nodes. It helps in parallel processing.
  - **Integration with other tools:** It is easier to integrate it with other tools like Apache Spark, Apache Flink, Apache Storm making it easier to build data processing pipelines.
  - **Cost effectiveness:** While many of the relational databases use expensive hardware, Hadoop use inexpensive hardware to do the parallel processing.
  - With all the above-mentioned characteristic of Hadoop, it plays crucial role in big data processing and analysis.

Dataset selection

- I chose to implement Inverted Index using Hadoop MapReduce on the datasets(datasets provided in data folder). It contains 18 text files. These text files contain any one of these like it might be a novel or a story or a poem or tragedie. The size of these files all together is ~11.1 MB. All files are less than MB except 1. 'The King James Bible'(bible-kjv) file is ~4 MB.
- **Text files contains follows topics:**
  - Emma by Jane Austen 1816
  - Persuasion by Jane Austen 1818
  - Sense and Sensibility by Jane Austen 1811
  - The King James Bible

- Poems by William Blake 1789
- Stories to Tell to Children by Sara Cone Bryant 1918
- The Adventures of Buster Bear by Thornton W. Burgess 1920
- Alice's Adventures in Wonderland by Lewis Carroll 1865
- The Ball and The Cross by G.K. Chesterton 1909
- The Wisdom of Father Brown by G. K. Chesterton 1914
- The Man Who Was Thursday by G. K. Chesterton 1908
- The Parent's Assistant, by Maria Edgeworth
- Moby Dick by Herman Melville 1851
- Paradise Lost by John Milton 1667
- The Tragedie of Julius Caesar by William Shakespeare 1599
- The Tragedie of Hamlet by William Shakespeare 1599
- The Tragedie of Macbeth by William Shakespeare 1603
- Leaves of Grass by Walt Whitman 1855

- **Dataset relevance to research objective and justification of why this dataset is suitable:**
  - The dataset reflects the real word scenario where documents can be unstructured and needs further processing.
  - The dataset includes different kinds of literature like novels, stories, poems, tragedies which offers diverse text formats.
  - The text documents are of varying sizes showing the effectiveness in processing and storing variety of data.
  - These text documents provide opportunity to explore optimization techniques in Hadoop distributed file systems for efficient data storage and retrieval due to varying sizes.
  - These text documents need to be further pre-processed which is one of the important aspects to evaluate MapReduce functionality.
  - These text documents are perfect example to demonstrate inverted index using Hadoop. Because Inverted index is a data structure that stores the mapping between word and its location, and these text documents is all about words.

Code Implementation

Workflow:

- The **mapper** class processes each document which is a whole text file without splitting the file. It reads the contents of the file and converts it into a string. It is also responsible for preprocessing including tokenization, converting to lower case, and removing punctuation. I also removed numbers. Since we were not asked about stop words, I didn't remove stop words. The mapper then checks for unique words using a HashSet and emits each unique word along with the document ID(I used the name of the text file as document ID) as the key-value pair. Since we were not asked about the count of word in a document, I just used unique words in a document.
- The **reducer** class takes the output from the mapper which is word and document ID. It processes each word and compiles a list of unique document IDs where this word appears. The reducer outputs the word as the key and a comma-separated list of document IDs as the value.
- The **runner** class configures and starts the Hadoop job. It sets up the job with the custom input format (WholeFileInputFormat), mapper, reducer, and specifies the output key and value types. It also defines the input and output paths for the job which are the arguments. It is the entry point to execute the MapReduce job.
- The **WholeFileInputFormat** class is a custom class to input format for reading each file as a single record. It overrides the isSplitable method to ensure that files are not split. Each mapper processes one entire text file as one split at a time. Without this class, mapper would just take document and split the document in lines. It uses a custom RecordReader (WholeFileRecordReader) to read the entire file content into a BytesWritable object along with the file name as the key.

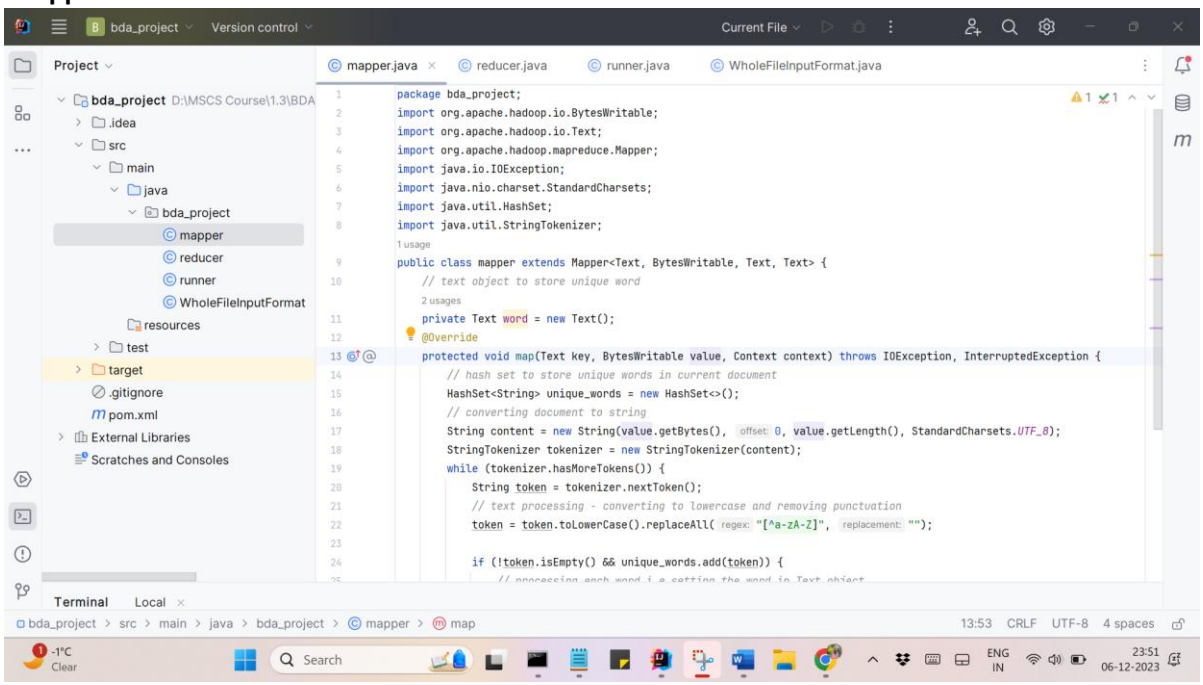
Execution of program:

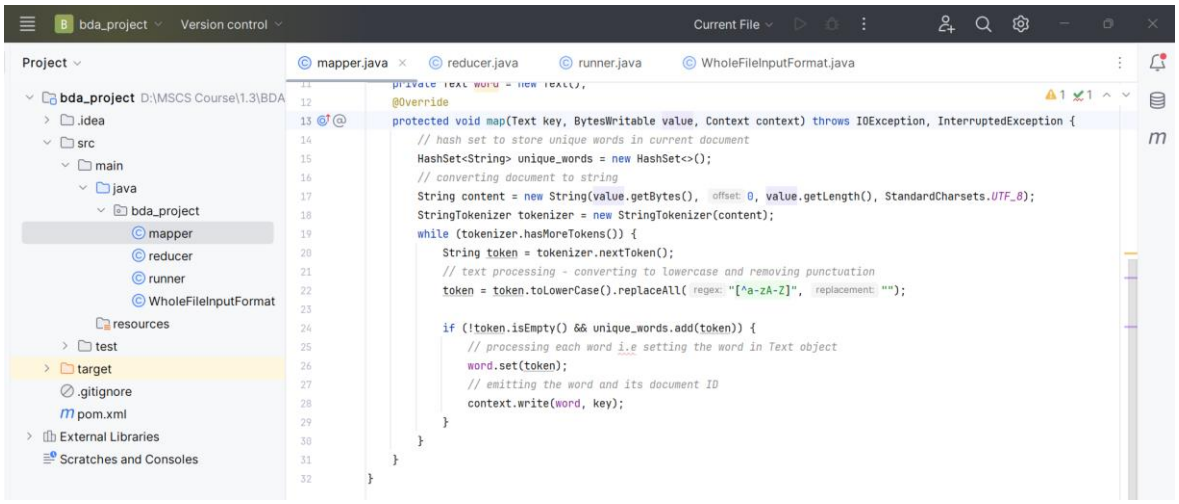
- **Initial setup:** First we create project folder in hdfs and upload the data folder give by professor.

```
Select Administrator: Command Prompt

C:\>cd hadoop-3.2.4

C:\hadoop-3.2.4>cd sbin

C:\hadoop-3.2.4\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
starting yarn daemons

C:\hadoop-3.2.4\sbin>hdfs dfs -mkdir /project

C:\hadoop-3.2.4\sbin>hadoop fs -copyFromLocal "D:\MSCS Course\1.3\BDA\Assignments\project\data" /project

C:\hadoop-3.2.4\sbin>hadoop fs -ls /project
Found 1 items
drwxr-xr-x   - HP supergroup          0 2023-12-05 19:24 /project/data

C:\hadoop-3.2.4\sbin>hadoop fs -ls /project/data
Found 19 items
-rw-r--r--   1 HP supergroup       6148 2023-12-05 19:24 /project/data/.DS_Store
-rw-r--r--   1 HP supergroup     887071 2023-12-05 19:24 /project/data/austen-emma.txt
-rw-r--r--   1 HP supergroup     466292 2023-12-05 19:24 /project/data/austen-persuasion.txt
-rw-r--r--   1 HP supergroup     673022 2023-12-05 19:24 /project/data/austen-sense.txt
-rw-r--r--   1 HP supergroup    4332554 2023-12-05 19:24 /project/data/bible-kjv.txt
-rw-r--r--   1 HP supergroup      38153 2023-12-05 19:24 /project/data/blake-poems.txt
-rw-r--r--   1 HP supergroup     249439 2023-12-05 19:24 /project/data/bryant-stories.txt
-rw-r--r--   1 HP supergroup      84663 2023-12-05 19:24 /project/data/burgess-busterbrown.txt
-rw-r--r--   1 HP supergroup     144395 2023-12-05 19:24 /project/data/carroll-alice.txt
-rw-r--r--   1 HP supergroup     457450 2023-12-05 19:24 /project/data/chesterton-ball.txt
-rw-r--r--   1 HP supergroup     406629 2023-12-05 19:24 /project/data/chesterton-brown.txt
-rw-r--r--   1 HP supergroup     320525 2023-12-05 19:24 /project/data/chesterton-thursday.txt
-rw-r--r--   1 HP supergroup     935158 2023-12-05 19:24 /project/data/edgeworth-parents.txt
-rw-r--r--   1 HP supergroup    1242990 2023-12-05 19:24 /project/data/melville-moby_dick.txt
-rw-r--r--   1 HP supergroup     468220 2023-12-05 19:24 /project/data/milton-paradise.txt
-rw-r--r--   1 HP supergroup     112310 2023-12-05 19:24 /project/data/shakespeare-caesar.txt
-rw-r--r--   1 HP supergroup     162881 2023-12-05 19:24 /project/data/shakespeare-hamlet.txt
-rw-r--r--   1 HP supergroup     100351 2023-12-05 19:24 /project/data/shakespeare-macbeth.txt
-rw-r--r--   1 HP supergroup     711215 2023-12-05 19:24 /project/data/whitman-leaves.txt

C:\hadoop-3.2.4\sbin>
```

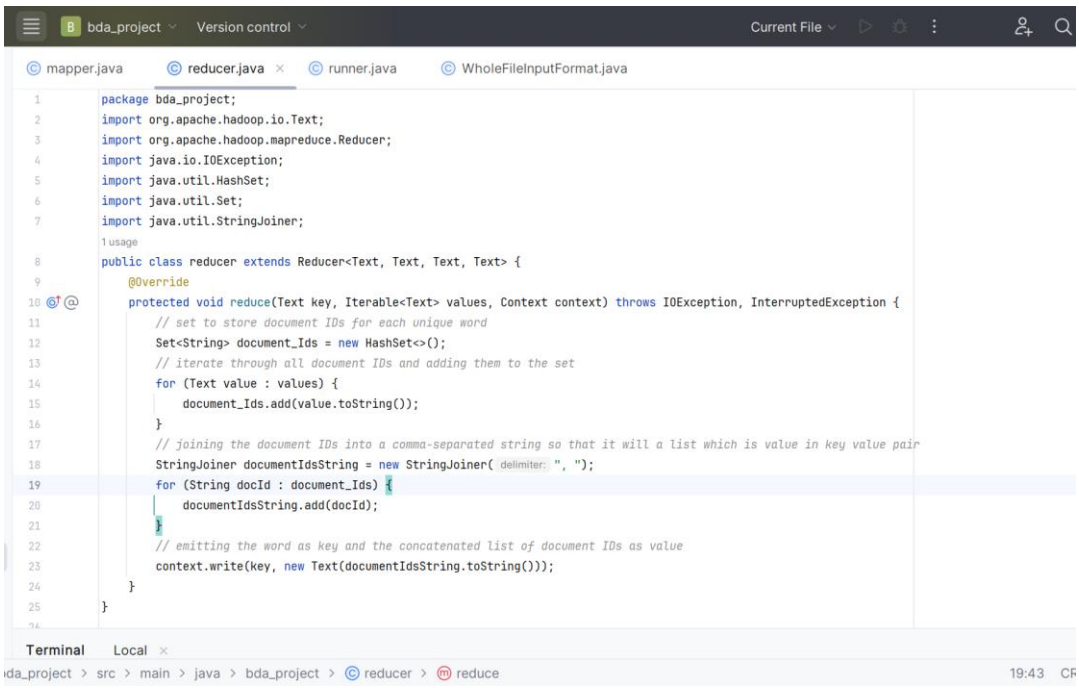
- Open IntelliJ IDEA IDE and write code for classes as below.

- **Mapper**

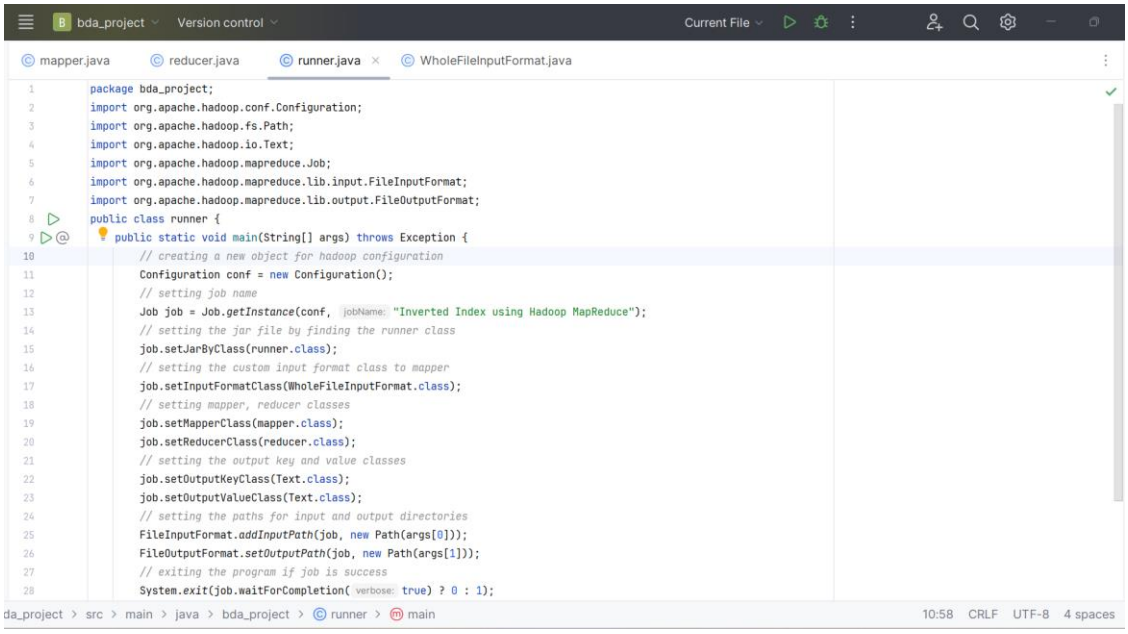




- **Reducer**



- **Runner**



- **WholeFileInputFormat**

bda_project > Version control > Current File

mapper.java | reducer.java | runner.java | WholeFileInputFormat.java

```java
package bda_project;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import java.io.IOException;
1 usage
public class WholeFileInputFormat extends FileInputFormat<Text, BytesWritable> {
    @Override
    protected boolean isSplitable(JobContext context, Path filename) {
        // ensuring each text file is not split and processing as a whole by a single mapper
        return false;
    }
    @Override
    public RecordReader<Text, BytesWritable> createRecordReader(InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException {
        // creating a new record reader
        WholeFileRecordReader reader = new WholeFileRecordReader();
        // initializing the record reader with the input split and context
        reader.initialize(split, context);
        return reader;
    }
```

bda_project > src > main > java > bda_project > WholeFileInputFormat 1:21 CRLF UTF-8 4 spaces

bda_project > Version control > Current File

mapper.java | reducer.java | runner.java | WholeFileInputFormat.java

```java
    public static class WholeFileRecordReader extends RecordReader<Text, BytesWritable> {
        3 usages
        private FileSplit fileSplit;
        2 usages
        private JobContext jobContext;
        2 usages
        private Text key;
        2 usages
        private BytesWritable value;
        3 usages
        private boolean processed = false;
        @Override
        public void initialize(InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException {
            // casting and storing the split as a FileSplit
            this.fileSplit = (FileSplit) split;
            // storing the job context
            this.jobContext = context;
        }
        @Override
        public boolean nextKeyValue() throws IOException, InterruptedException {
            if (!processed) {
                // buffer to store file contents
                byte[] contents = new byte[(int) fileSplit.getLength()];
                // getting the path of the file to be read
                Path file = fileSplit.getPath();
                // getting the FileSystem object
                FileSystem fs = file.getFileSystem(jobContext.getConfiguration());
                try (FSDataInputStream in = fs.open(file)) {
```

bda_project > src > main > java > bda_project > WholeFileInputFormat 1:21 CRLF UTF-8 4 spaces

bda_project > Version control > Current File

mapper.java | reducer.java | runner.java | WholeFileInputFormat.java

```java
                    in.readFully(contents);
                }
                // setting the key as the file name and value as the file contents and marking this file as processed
                key = new Text(file.getName());
                value = new BytesWritable(contents);
                processed = true;
                return true;
            }
            return false;
        }
        @Override
        public Text getCurrentKey() throws IOException, InterruptedException {
            // returning the current key (file name)
            return key;
        }
        @Override
        public BytesWritable getCurrentValue() throws IOException, InterruptedException {
            // returning the current value (file contents)
            return value;
        }
        @Override
        public float getProgress() throws IOException {
            // returning the progress of the record reader
            return processed ? 1.0f : 0.0f;
        }
        @Override
        public void close() throws IOException {
        }
```

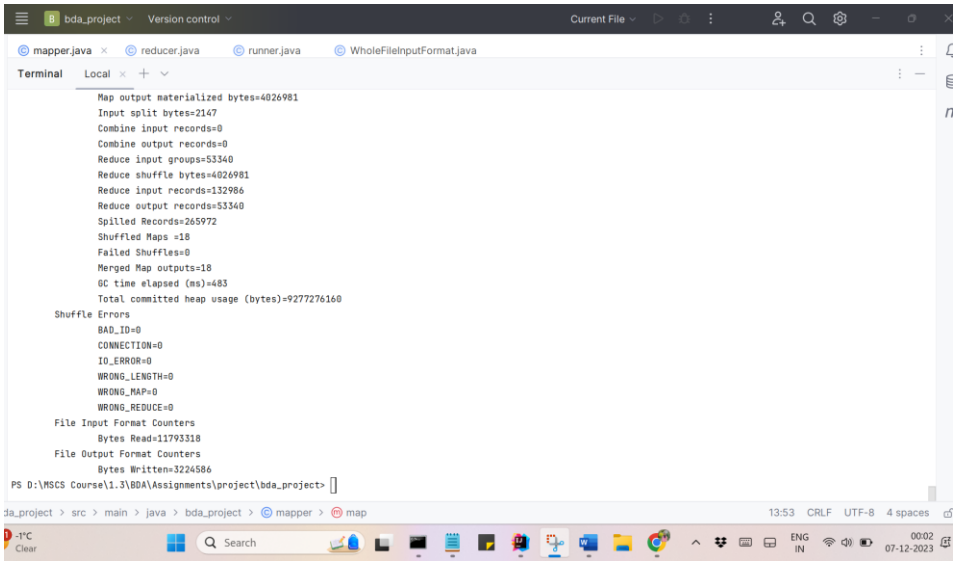
bda_project > src > main > java > bda_project > WholeFileInputFormat > WholeFileRecordReader > nextKeyValue 57:34 CRLF UTF-8 4 spaces

Result Analysis:

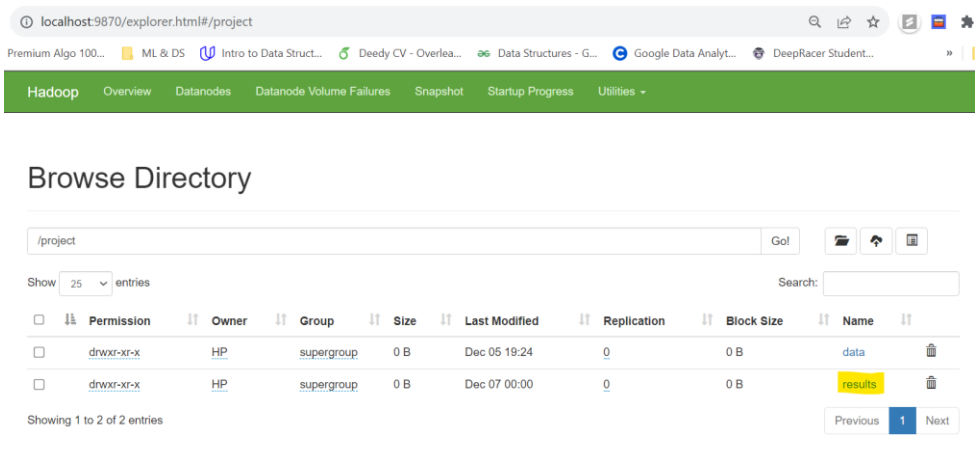
- Once we complete coding part, we do maven clean and install which generates the jar file.
- Jar file generated: bda_project-1.0-SNAPSHOT.jar
- Now run the Hadoop job using the following cmd which generates the key values pairs as intended that are stored in /project/results folder in hdfs.
- **Cmd to execute the program :**
- 'hadoop jar target/bda_project-1.0-SNAPSHOT.jar bda_project.runner /project/data /project/results'
- **Program execution:**

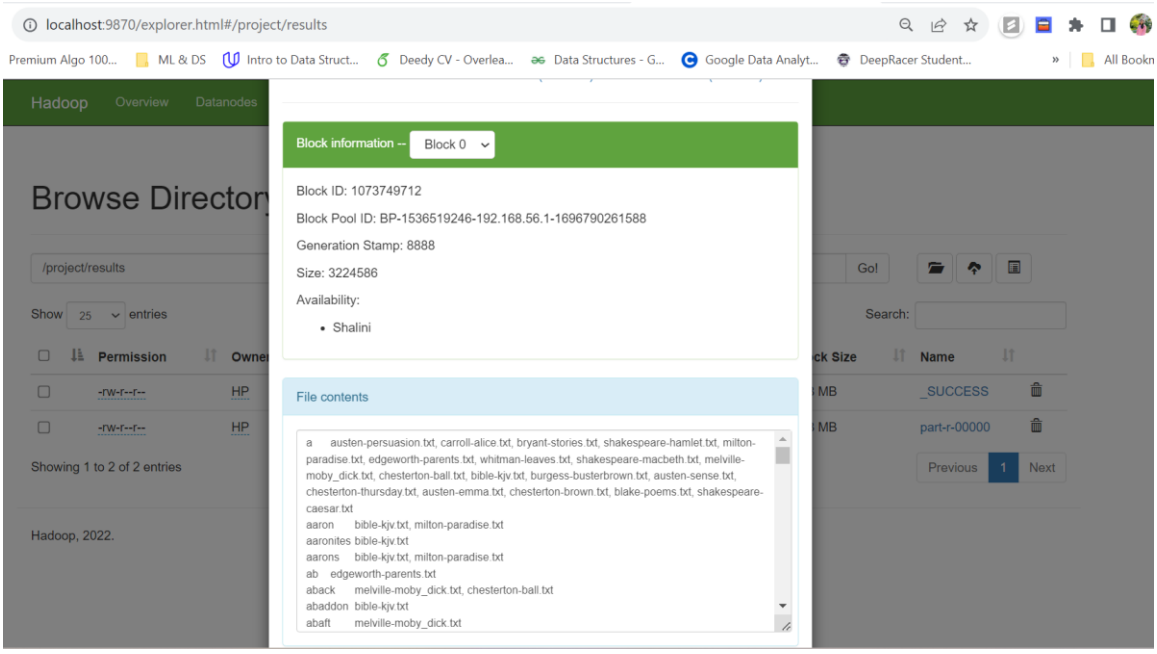
```
PS D:\MSCS Course\1.3\BDA\Assignments\project\bda_project> hadoop jar target/bda_project-1.0-SNAPSHOT.jar bda_project.runner /project/data /project/results
2023-12-06 23:59:58,433 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2023-12-06 23:59:58,520 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2023-12-06 23:59:58,520 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2023-12-06 23:59:59,093 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application w
ith ToolRunner to remedy this.
2023-12-06 23:59:59,240 INFO input.FileInputFormat: Total input files to process : 18
2023-12-06 23:59:59,261 INFO mapreduce.JobSubmitter: number of splits:18
2023-12-06 23:59:59,363 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local2102078911_0001
2023-12-06 23:59:59,364 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-12-06 23:59:59,556 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2023-12-06 23:59:59,568 INFO mapreduce.Job: Running job: job_local2102078911_0001
2023-12-06 23:59:59,570 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2023-12-06 23:59:59,583 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-12-06 23:59:59,583 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2023-12-06 23:59:59,584 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2023-12-06 23:59:59,698 INFO mapred.LocalJobRunner: Waiting for map tasks
2023-12-06 23:59:59,699 INFO mapred.LocalJobRunner: Starting task: attempt_local2102078911_0001_m_000000_0
2023-12-06 23:59:59,730 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-12-06 23:59:59,730 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2023-12-06 23:59:59,742 INFO util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on Linux.
2023-12-06 23:59:59,800 INFO mapred.Task:  Using ResourceCalculatorProcessTree : org.apache.hadoop.yarn.util.WindowsBasedProcessTree@74cafdaa
2023-12-06 23:59:59,820 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/project/data/bible-kjv.txt:0+4332554
```

- It prints a lot of informational messages, but I attached screenshots of cmd and end messages.



- **Results obtained from MapReduce program:**
  - The results file is generated in hdfs. I have attached the results file in submission zip file which contains key value pairs.





- ‘a’ is the most common word. It occurs in every document. So, for key word ‘a’, value is the name of every text file in data folder.

Conclusion

- To conclude, I have implemented inverted index using Hadoop MapReduce programming model. I have pre-processed the data in Mapper phase. My mapper output is unique word and document ID. Reducer takes output from mapper and gives unique word and the list of documents where this word is located. This project showcases how hadoop can process large datasets efficiently and in less time. I got results is less than a minute which is difficult to achieve with traditional programming algorithms. I have highlighted the significance of Hadoop in big data processing and analytics. Also, I have listed the challenges we face when optimizing data storage and retrieval in HDFS and solution to optimize the performance. These optimization strategies must be employed to ensure the performance especially in real world scenarios. Future work includes trying different indexing schemes on large datasets and comparing the performance.

References

[1] https://livebook.manning.com/book/hadoop-in-practice-second-edition/chapter-4/1
[2] https://ieeexplore.ieee.org/document/7924666
[3] https://ieeexplore.ieee.org/document/7073217
[4] https://www.linkedin.com/pulse/how-optimize-hdfs-performance-large-scale-data-soumyadeep-mandal/
[5] https://www.linkedin.com/advice/3/what-best-practices-tuning-hdfs-performance-skills-big-data?trk=article-ssr-frontend-x-article_more-articles_related-content-card
[6] https://www.itm-conferences.org/articles/itmconf/pdf/2023/03/itmconf_icdsia2023_03001.pdf
[7] https://www.geeksforgeeks.org/hadoop-features-of-hadoop-which-makes-it-popular/
[8] https://www.datasciencecentral.com/importance-of-big-data-analytics-tool-hadoop/