Assignment 2: Implementing MapReduce

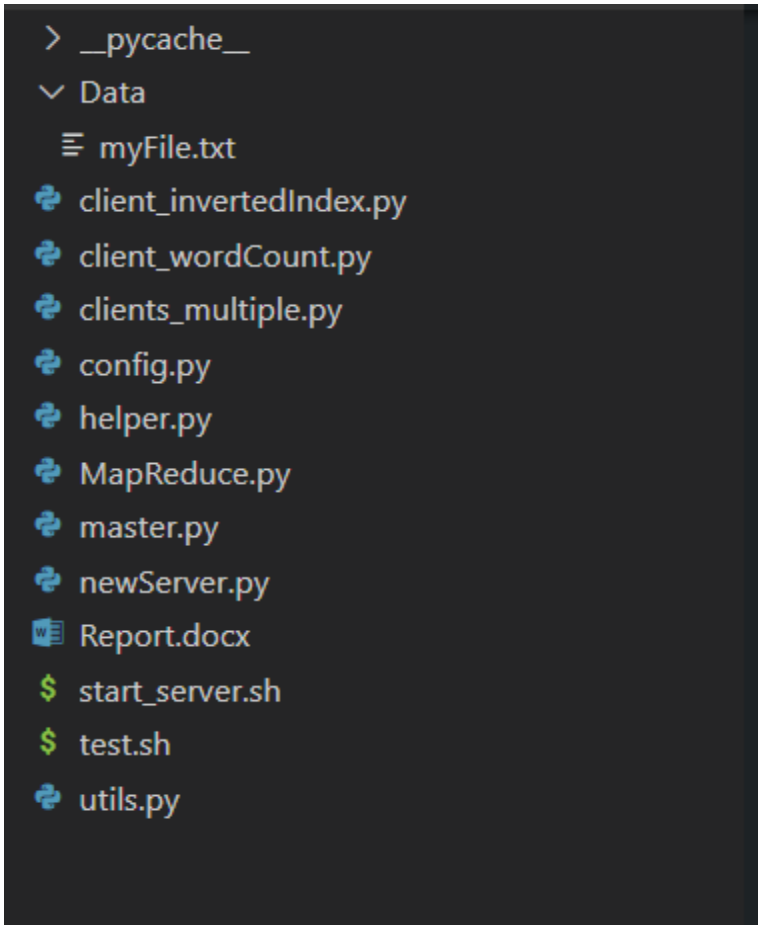
INTRODUCTION:

MapReduce is a programming methodology for processing huge data collections. A map function analyzes a key/value pair to yield a collection of intermediate key/value pairs, whereas a reduce function merges all intermediate values associated with the same intermediate key. This model may be used to represent a wide range of real-world operations.

In this assignment, the two main applications are:

1. Word Count
2. Inverted Index

ARCHITECTURE:



The implementation has the following files:

SUBMITTED BY:
PRASHASTI KARLEKAR
prkarl@iu.edu

Assignment 2: Implementing MapReduce

1. **SERVER:**
   A TCP server written in Python 3 that receives client requests and performs map reduce operations based on the provided input file. This server is capable of handling connections from several clients at the same time (the response time majorly depends on the capability of the hardware its running on). The server creates and calls an object of main, which is located in the Master implementation. The result of the main is returned to the client by the server.

2. **MASTER:**
   This acts as the master node. Based upon the request by the client, master spawns the respective map and reduce functions on the input data located at the input location provided by the client. If the map function is "word_count", the master will create an object of class WordCount which then handles further operations. If it receives "inverted_index", it will create an object of InvertedIndex class.

3. **MAPREDUCE**:
   This contains the implementations of WordCount and InvertedIndex classes.
   - WordCount class implementation:

     The mapper tasks reads the input file corresponding to the respective file split. It then emits key-value pairs from the file and passes this to the combiner. The combiner reformats the produced list by the mapper function into key, value pairs that complements as an input to the reduced function by combining all the values corresponding to each key. It returns tuples of (word, [# of times the word appears]). This combined data is saved on the local file system. The reduce function then reduces the computed results by each worker into the final output by summing all the values.

   - InvertedIndex class implementation:

     The first task performed on the input is the split_input operation where the file is divided into chunks equal to the number of mappers. The mappers run parallelly with each mapper getting a chunk and the output of each mapper is combined at the end and stored as MergedData. After this, reducer function gets a unique key and performs reduce operation on it. Once all the reducers have been executed, we combine the results and store it as an output file at the output location provided.

4. **UTILS:**
   This file contains the helper functions required during the map and reduce operations. These include split_input, merge_files and join_files.

5. **HELPER :**

   This includes the map and reduce functions corresponding to WordCount and InvertedIndex objects.

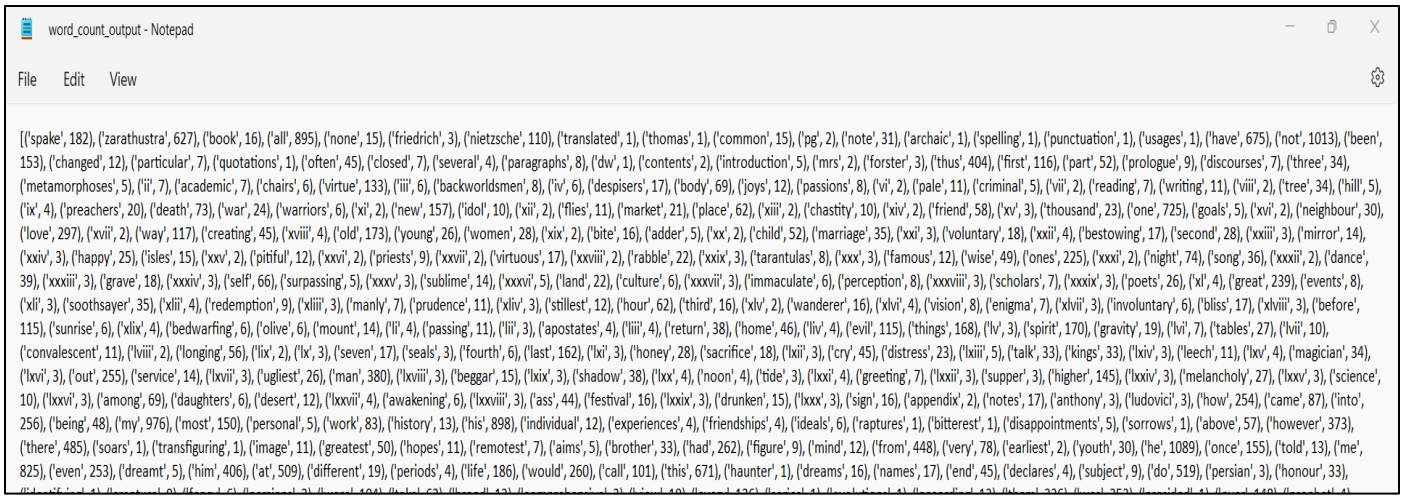
6. **CONFIG:**

   This file contains Host and port information which can be changed accordingly.

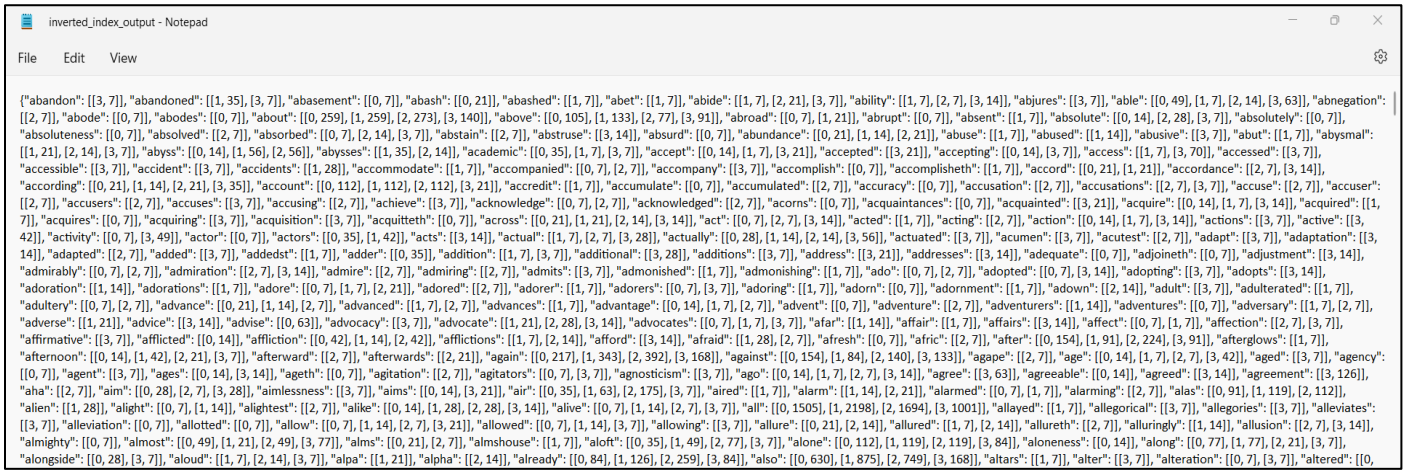
SUBMITTED BY:
PRASHASTI KARLEKAR
prkarl@iu.edu

Assignment 2: Implementing MapReduce

Below are the output files for both the applications:



Output for Word Count



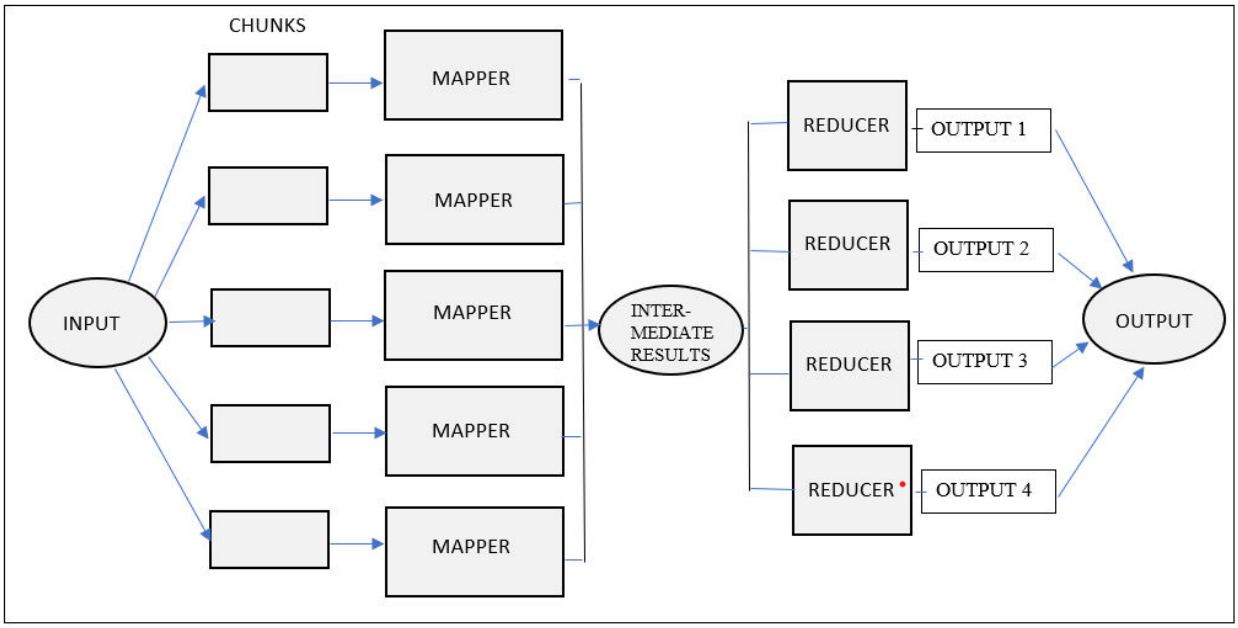
Output for Inverted Index

FLOW CHART:

This flow chart describes the working of the Map Reduce model that has been implemented.

SUBMITTED BY:
PRASHASTI KARLEKAR
prkarl@iu.edu

Assignment 2: Implementing MapReduce



BONUS COMPONENTS:

- Protocol Buffers are implemented as a bonus component, since all the operations adhere to the same protocol with which intermediate data is passed to reducers. This leads to an efficient implementation of reducers.
- Fault Tolerance has been implemented as a bonus component. If the master fails, the client can resend the request again. If the map function or the reducer function fails, they get restarted again. In case of reducer function, it will simply read the stored intermediate data.

TESTS:

To start the server, run :

./start_server.sh

To run the test files, run the below script as follows:

./test.sh

There are three test cases in these files:

1. client_invertedIndex: This file passes input to run inverted index application on the input file.

```
-------TEST CASE TO RUN INVERTED INDEX APPLICATION-------

Response from the server
Server -- Task comepleted.
```

SUBMITTED BY:
PRASHASTI KARLEKAR
prkarl@iu.edu

Assignment 2: Implementing MapReduce

2. client_word_count: This file passes input to run word count application on the input file.

```
-------TEST CASE TO RUN WORD COUNT APPLICATION-------

Server response:
Server -- Task completed.
```

3. clients_multiple: This file tests if the server is able to handle multiple clients parallelly.

```
-------TEST CASE TO CONNECT MULTIPLE CLIENTS TO THE SERVER-------

Connecting to Client 1
Connecting to Client 2
Response from Client 1:
Server -- Task completed.
Response from Client 2:
Server -- Task comepleted.
```

4. Below is the result of running the ./start_server.sh and ./test.sh command:

```
prashasti@DESKTOP-CD2K870:/mnt/c/STUDY/EDS/OldMapReduce4.0$ ./test.sh
-------TEST CASE TO RUN WORD COUNT APPLICATION-------

Server response:
Server -- Task completed.
-------TEST CASE TO RUN INVERTED INDEX APPLICATION-------

Response from the server
Server -- Task completed.
-------TEST CASE TO CONNECT MULTIPLE CLIENTS TO THE SERVER-------

Connecting to Client 1
Connecting to Client 2
Response from Client 1:
Server -- Task completed.
-----------Execution of Client 1 request is completed.-----------
Response from Client 2:
Server -- Task completed.
-----------Execution of Client 2 request is completed.-----------
prashasti@DESKTOP-CD2K870:/mnt/c/STUDY/EDS/OldMapReduce4.0$
```

Output for ./test.sh

SUBMITTED BY:
PRASHASTI KARLEKAR
prkarl@iu.edu

Assignment 2: Implementing MapReduce

```
prashasti@DESKTOP-CD2K87O:/mnt/c/STUDY/EDS/OldMapReduce4.0$ ./start_server.sh
Waiting for a Connection..
Connected to: 127.0.0.1:53518
Thread Number: 1
SERVER -- PROCESSING THREAD
SERVER ----Task completed
Connected to: 127.0.0.1:53520
Thread Number: 2
SERVER -- PROCESSING THREAD
SERVER ----Task completed
Connected to: 127.0.0.1:53522
Thread Number: 3
SERVER -- PROCESSING THREAD
Connected to: 127.0.0.1:53524
Thread Number: 4
SERVER -- PROCESSING THREAD
SERVER ----Task completed
SERVER ----Task completed
```

Output for ./start_server.sh

LIMITATIONS:

- The application does not any other applications of map reduce and only includes Word Count and Inverted Index.
- The application uses local file system as a datastore and does not work on a distributed file system.
- The number of clients connected to the server at the same time is restricted by the hardware settings on which the server will be executed.

FUTURE SCOPE:

- Some nodes are slow in terms of execution. A correct model with load balancing implementation can be applied.
- A distributed datastore can be implemented within the application which will allow it to run in a more scalable environment and hence improving the performance.

SUBMITTED BY:
PRASHASTI KARLEKAR
prkarl@iu.edu