Name: Aditya Arun Rudrawar
Assignment: Map Reduce

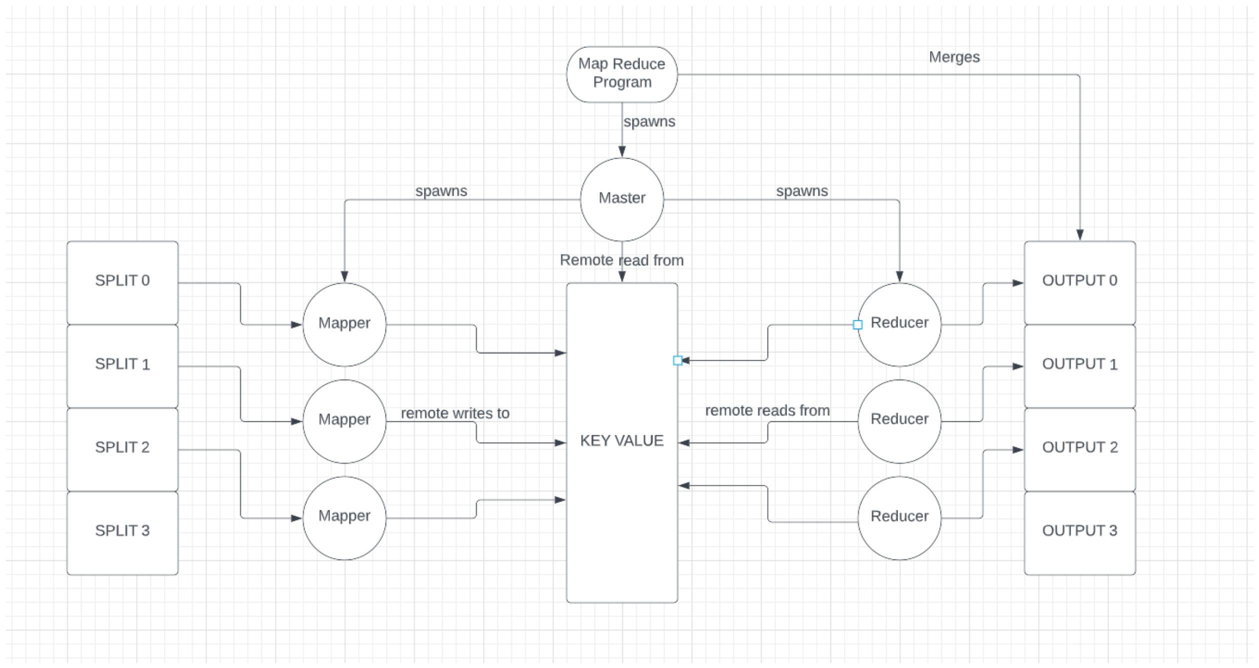
How to run

1. Installed the requirements from the requirements.txt
2. Change the .env according to your system.
    a. Change the HOST, PORT according to your machine for the key value server.
    b. Change the Key value storage path to a folder.
3. Start the key value server by using the command: **python server.py**
4. The input for mapreduce is stored inside the Input_files folder.
5. Multiples examples of mapreduce are given
    a. Wordcount1.py - Simple file
    b. Wordcount2.py - Texas book
    c. Wordcount3.py - Multiple files as input
    d. Invertedindex.py - example of inverted index.
6. Run each file, using the command **python <filename.py>**
7. After processing mapreduce will generate an output file stored in the output directory.
8. The Map reduce implementation takes these arguments:

mapperFunction	Map function given by the user
reducerFunction	Reduce function given by the user
numberOfReducers	Number of reducers
splitSize	Size of each piece of an input file
outputFileName	Name of the output File that the user wants, defaults to “”
outputFileDirectory	Folder where the output will be stored
inputFilesLocation	An array of location of the input files.

Design

1. It uses a shared memory database for communication. [BONUS] keyvalue
2. Only using network-based communication.
3. File split is based on the size mentioned by the user, it takes the size and divides the file into chunk of almost that size. It makes sure it is not cutting any word in half, wether it be ascii or unicode, so that case is handled.



IMPLEMENTATION

EXECUTION OVERVIEW

1. The program takes the map and reduce functions from the user, the size of a block each reducer can process. It also takes certain other things such as the output file location, input file location as an array, it can have multiple files in the input array.
2. The MapReduce library first splits the input files into chunks of data, the size of the chunk is also an input from the user. Each split acts as a separate mapper job. Each job is indexed and stored in the key value storage. Each mapper job has a file location, and start, end offset pointers, these 3 things represent a chunk.
3. MapReduce then spawns a Master Node, passing the mapper jobs and other information such as the number of reducers.
4. Master starts the mappers, the number of mappers running at a time are fixed, and new mapper jobs are started when previous mappers are completed running.
5. Each mapper runs the input function provided by the user and updates its status in the key value storage, this status is accessed by the master through a network call.
6. After completing its task, the mapper writes its output to the key value store. Each output is associated with a unique id. The master has access to each output key,
7. After completing all the mapper tasks, the master starts the reducer phase. It spawns the number of reducers and passes the location of each mapper output to every reducer.
8. Each reducer loads all the mapper output and parses the keys; if the hash of the key is the same as the reducer id, it processes that input. This is how the Groupby and shuffle is implemented.
9. Atlast, output of each reducer is then merged into one file for the final output.

TESTING:

1. The map reduce implementation is tested against all the input files given in the zip folder.
2. Samplecount.txt contains simple paragraphs.
3. Five_years_in_Texas and gutenberg.txt contains unicode characters.
4. The output of all the given examples is in the output folder.
5. The inverted index is tested by passing the appropriate map reduce function. The example is given under the file name invertedindex.py

Commands to run each file

Key value server	python server,py
Wordcount test 1	python wordcount1.py
Wordcount test 2	python wordcount2.py
Wordcount test 3	python wordcount3.py
Wordcount test 4	python wordcount4.py
Inverted index test 1	python invertedindex.py

BONUS:

1. Used key value store.
2. Implemented Fault tolerance by restarting the task of a mapper/reducer.
3. Can parse unicode characters.

Limitations and Assumption

1. If the master goes down, the user needs to start the mapreduce all over again.
2. The map and reduce functions should not change. They are only supported when the return values are only two.
3. Assumes inverted index means a word with all the different locations it exists at. Does not concern the offsets within each file.
4. Key value storage is a single point failure.
5. Some performance measures as well.