

## **A Non-MapReduce like MapReduce Prototype**

**A Report by**

Glenn Healy

SR017409

[g.healy@student.reading.ac.uk](mailto:g.healy@student.reading.ac.uk)

## Introduction

The goal for this project was to develop a basic prototype implementation of the behaviours from Hadoops MapReduce project. And be able to read in a series of data from a CSV file and carry out some basic data validation and correction and then sort the data and carry out three specific operations on the data to generate smaller and more manageable data sets.





## Development / MapReduce comparison

For this, I chose to make use of Java, and developed my prototype using the java libraries. Development comprised of several basic stages, each with their own challenge and requirements:

1. Design and abstract the overall behaviours
2. Implement a basic data to object design
3. Read the file in, separating the data by the delimiter and passing the data to a array
4. Transfer this to a hashmap bound to the required key
5. Shuffle the data as needed
6. Reduce the data to smaller sets and output the data to csv files

I then went about making sure I got the basic building locks of the program designed, and broke up the program into what I guessed out be the number of classes I would have needed, which ended up being:

1. A Mapper Class (also contains the main function of the program)
2. A FlighData Class (contains the object information for the data design)
3. A Validation Class (used to call the validation functions)
4. A Reducer class (contains the reducers)

 Mapper.class	Today, 00:03	5 KB	Java class file
 Reducer1.class	Yesterday, 23:07	3 KB	Java class file
 FlightData.class	Yesterday, 22:25	2 KB	Java class file
 Validation.class	Yesterday, 22:25	1 KB	Java class file

I developed the program a single stage as a time as per the guide on blackboard, I ensured that I could accurately read in the data, verify the integrity of that data and then map it to a key value pair map and shuffle it around as needed. This is in comparison to how MapReduce does it. The only major failure in my program are my Reducer functions, where I wouldn't quite wrap my head around how I was supposed to be passing the data to the reducer and thus, outputting it to files.

For this project, I attempted to recreate the very and most basic behaviour of MapReduce. This comprises of the following behaviours:

1. Data loading and reading – read in a vile, and pass the data to an array
2. Parse the data to a hashmap carrying out data validation to ensure it meets the requirements (no missing data etc)
3. Shuffle the data and map it to a series of key value pairs
4. Reduce the data sets into smaller parts and output reports on this.

To do this I first got the basic CSV file reading working:

```
//file reading - here.
System.out.println("Please enter a file directors and name (e.g. /documents/data/data.csv");
Scanner filePathInput = new Scanner(System.in);
String filePath = filePathInput.nextLine();
File file = new File(filePath);

//file checking
if (file.isFile()) {
    Scanner fileInput = null;
    try {
        fileInput = new Scanner(file);
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return;
    }
}
```

I then developed a way to pass this into the arrays correctly by passing the data to an object instance of my data class:

```
public class FlightData {

    //variables to be assigned to each object type.

    private String passengerID;
    private String flightID;
    private char[] fromID = new char[3];
    private char[] tooID = new char[3];
    public Date departTime;
    public long flightTimeMins;
    public Date arrivalTime;

    //Constructor:
    public FlightData(String passengerID, String flightID, char[] fromID, char[] tooID, Date departTime, long flightTimeMins) {
        //data objects (elements in the file)
        setPassengerID(passengerID);
        setFlightID(flightID);
        setFromID(fromID);
        setTooID(tooID);
        setFlightTimeMins(flightTimeMins);
        setDepartTime(departTime);
        setArrivalTime(arrivalTime);
    }
}
```

```
ArrayList<String> lineBuffer = new ArrayList<>();
ArrayList<String> errorBuffer = new ArrayList<>();

//looping through the file line by line
while (fileInput.hasNextLine()) {
    String line = fileInput.nextLine();
    if (line.matches("[A-Z]{3}[0-9]{4}[A-Z]{2}[0-9]{1}[\\,]{1}[A-Z]{3}[0-9]{4}[A-Z]{1}[\\,]{1}[A-Z]{3}[\\,]{1}[A-Z]{3}[\\,]{1}[0-9]{10}[\\,]{1}[0-9]{1,4}")) { //regex to
        lineBuffer.add(line);
    } else {
        errorBuffer.add(line);
        System.out.println("error found at line:" + errorBuffer); //was using to check buffer error working.
        File errors = new File("errors.log");
        FileWriter fw = null;
        try {
            fw = new FileWriter(errors);
        } catch (IOException e) {
            e.printStackTrace();
        }
        PrintWriter epw = new PrintWriter(fw);
        epw.append(line);
        epw.close();
    }

    HashMap<String, ArrayList<FlightData>> test = mapper(lineBuffer);
    ArrayList<String> keys = new ArrayList<String>(test.keySet());
    for (int x = 0; x < keys.size(); x++) {
        System.out.println(reducer(keys.get(x), test.get(keys.get(x))));
    }
    // System.out.println(errorBuffer);
}
```

This is also where I carried out my data validation via a slightly complicated but also very basic Regular expression, that is called and compared against as the file read is carried out. It checks against each line to ensure it is of the correct format:

```
while (fileInput.hasNextLine()) {
    String line = fileInput.nextLine();

    if (line.matches("[A-Z]{3}[0-9]{4}[A-Z]{2}[0-9]{1}[\\,]{1}[A-Z]{3}[0-9]{4}[A-Z]{1}[\\,]{1}[A-Z]{3}[\\,]{1}[A-Z]{3}[\\,]{1}[0-9]{10}[\\,]{1}[0-9]{1,4}")) { //regex
        to validate the line is not empty. && meets the basic requirements.
        lineBuffer.add(line);
    } else {
        errorBuffer.add(line);
        System.out.println("error found at line:" + errorBuffer); //was using to check buffer error working.
    }
}
```

If the line is not in the correct format or is two short, long etc. then is placed into the following array:

```
ArrayList<String> errorBuffer = new ArrayList<>();
```

Before then being printed to a file:

```
errorBuffer.add(line);
System.out.println("error found at line:" + errorBuffer); //was using to check buffer error working.
File errors = new File("errors.log");
FileWriter EW = null;
try {
    EW = new FileWriter(errors);
} catch (IOException x){
    x.printStackTrace();
}
PrintWriter epw = new PrintWriter(EW);
epw.append(line);
epw.close();
```

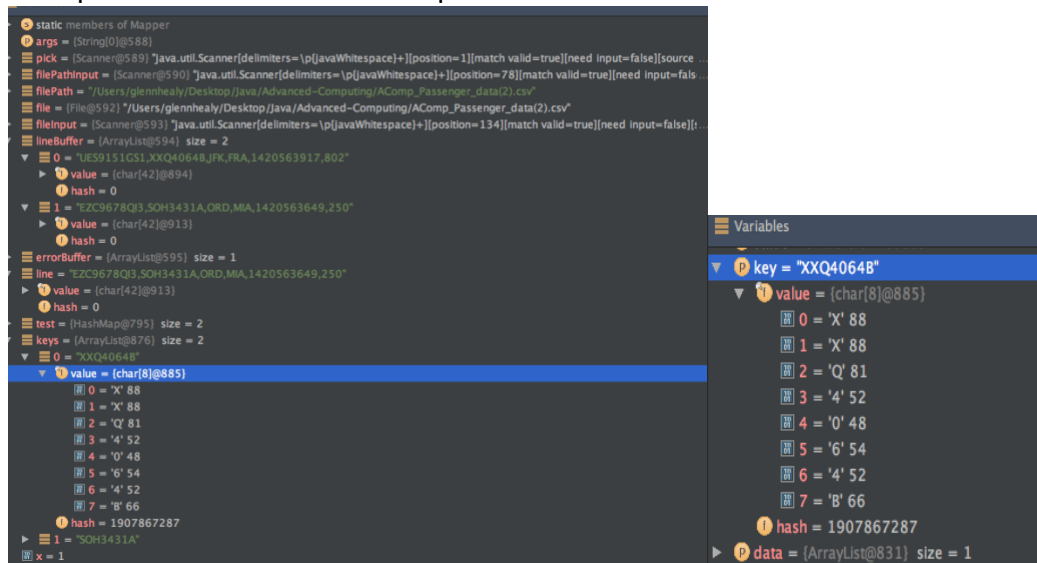
From here, I then carry out the mapping functions, where I map each element in the current array into a Hashmap based on key value pairs. Where the key for the hash is the FlightID where each flightID key is lined to many passengers, two airports and departure times and arrival times:

```
//Mapper function
public static HashMap<String, ArrayList<FlightData>> mapper(ArrayList<String> lineBuffer) {

    HashMap<String, ArrayList<FlightData>> mapdata = new HashMap<>(); //array list for Mapdata object
    for (String flightData : lineBuffer) {
        String[] str = flightData.split(",");
        FlightData flight = new FlightData(str[0], str[1], str[2].toCharArray(), str[3].toCharArray(), new Date(Long.valueOf(str[4])),
        Long.valueOf(str[5]).longValue()); //creating the object
        mapdata.get(flight.getFlightID()); //getting the flight data
        if (mapdata.containsKey(flight.getFlightID())) { //checking if the data for the object contains hash key Flightdata
            mapdata.get(flight.getFlightID()).add(flight);
        }
        else if (mapdata.containsKey(flight.getFromID())) { //for airport ID (from - to)
            mapdata.get(flight.getFromID()).add(flight);
            ArrayList<FlightData> noID2 = new ArrayList<>(); //creating an array for noID
            noID2.add(flight);
            mapdata.put(flight.getFlightID(), noID2);
        }
        else { //for data not assigned to a key
            ArrayList<FlightData> noID = new ArrayList<>(); //creating an array for noID
            noID.add(flight);
            mapdata.put(flight.getFlightID(), noID);
        }
    }

    return mapdata;
}
```

It also splits the data from the array at the delimiter “,” and assigned an individual value for each piece of data in the hashmap:



From here I developed my Reducer functions. For the reducers, I decided to have them in a separate class and simply call the relevant reducer to carry out the specific job needed via function call:

```
//calls the reducer class 1 for each job.
public static String reducer(String key, ArrayList<FlightData> data) {
    // mymenu();
    String output = "";
    Reducer1 pickreducer = new Reducer1();

    if (job == 1) {
        Reducer1.reducerOne(key, data);
    } else if (job == 2) {
        Reducer1.reducerTwo(key, data);
    } else
        Reducer1.reducerThree(key, data);
    // System.out.println(job);

    return output;
}
```

This simply calls the relevant reducer and carries out jobs, for example, number of passengers per flight:

```
//number of passengers per flight
public static String reducerOne(String key, ArrayList<FlightData> data) {
    String output = "";
    for (int x = 0; x < data.size(); x++) {

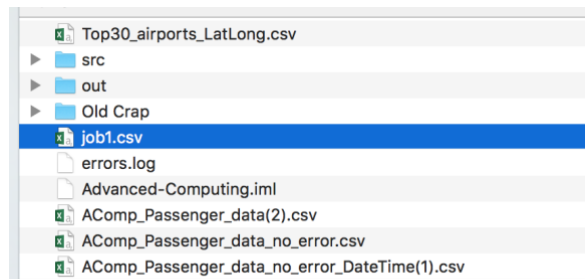
        output += "Flight " + key + " Has " + data.size() + " Passengers on boarded " + "\n";
    }
    System.out.println(output);
}
```

Where the output is accordingly, this is then output to a file:

```

Flight WPW9201U Has 10 Passengers on boarded
Flight DAU2617A Has 12 Passengers on boarded
Flight ULZ8130D Has 26 Passengers on boarded
Flight YZ04444S Has 15 Passengers on boarded
Flight PNE8178S Has 1 Passengers on boarded
Flight VDCP164W Has 1 Passengers on boarded

```



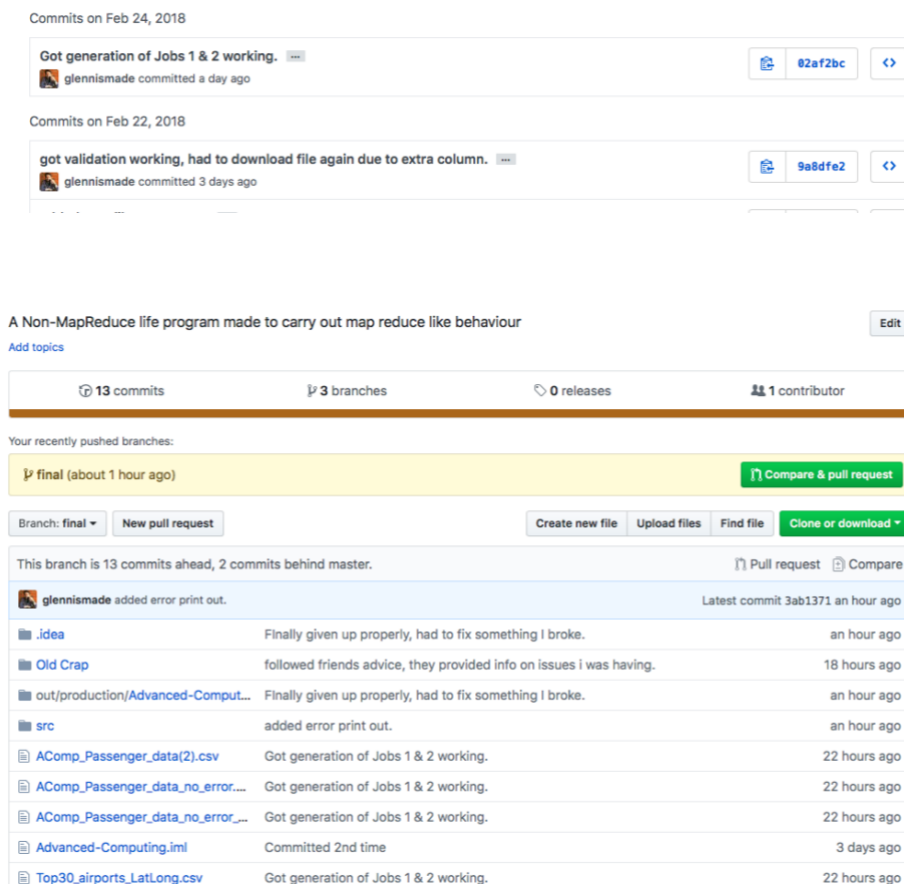
## Data Output formatting/reports

The data goes through the mapper function I developed, then gets passed to my reducers and output in the desired format.

For this, I carry out a series of data operations and formatting of the strings before outputting them to the console and then simply write them to a .csv file with a new line per data entry. Any errors that the program finds with the data are placed into a errors.log file

## Subversion Control

For the version control of the project, I made use of Git, Git Command line tools and my personal GitHub account and IntelliJ's built in version control tools to carry out a series of commits to my GitHub account across three branches all under 1 project as can be seen here:



## Reflections/Self-Appraisal

Throughout this project, I struggled to understand the behaviour of MapReduce properly and really had to grapple with the concepts and how each component was interacting with one another. I also struggled to really grasp how my implementation was intended to work.

This meant that while I followed the step by step guide provided by Atta on Blackboard and the lecture notes along with several videos and Stackoverflow posts to really try and get a grasp of how the behaviours I was implementing were supposed to work and how would be best to handle the data and where it was supposed to be going.

As such, I encountered several issues during development and found it hard to carry out the Reducers appropriately, resulting in the program being unfinished and only successfully completing the Final job request of Number of Passengers per flight. I also partially got the number of flights per airport working, however I had a hard time to really get this working as desired, resulting in this being incomplete.

This also meant that due to some issues and limitations in my design and implementation I was unable to get the 2<sup>nd</sup> job working of having a list of flights with their relevant details provided along with the flight ID.

This clearly is represented in my code and is evident in both the reducer design and the overall execution of parts of my code. I was struggling to get the program to work, resulting in me needing to correct several errors and bugs which prevented the program from running and meant that I had to make some concessions to get the program working resulting in my program not working as desired.

There is also an interesting bug I couldn't resolve where, the 2<sup>nd</sup> job will overwrite the data written to the csv each time it writes a line, even though I use the .append method. I tried to work on this, however was unable to correct this. The same thing also happens to the error.log when I call the 2<sup>nd</sup> job.

Given more time, and in hindsight I would go back to the drawing board and start again from scratch and change much of my design, especially my separating out my mapper into three separate mappers and ensure a more robust method for converting the data into hashmaps. I would also have added a GUI to allow for the data to be more easily viewed along with a file loader (file search).

Overall, I feel that the project was not as much of a success as I would have liked and was only able to get 1 job completely working and half of a 2<sup>nd</sup> one partially working. The program also has many problems and is not as well designed as I would have liked.

However, I am happy with my attempt, as this stretched my ability very far and showed me that I need to spend more time designing my programs in the future and need to work on my OOP concepts more.

## Conclusions

Overall, I feel that the development of the prototype did not go as well as I would have liked and resulted in me not really being able to complete the development to a standard I am happy with and means that the program is technically incomplete.

In hindsight I would do things very differently, but I am happy with what I have done overall, as I have never done anything remotely like this before.