# Non-MapReduce Prototype

**Jules Gribble - 23004709**
**BSc Computer Science with Industrial Year, j.a.gribble@student.reading.ac.uk**

## ABSTRACT

Big Data is quickly growing with more and more data created everyday. Using traditional data processing it is near impossible to get any useful data. This report shows how programing models such as MapReduce can analyse data successfully giving a useful output. The report walks through the assignment including the objectives, design and implementation.

The design discuss what each part of MapReduce does, including the expected inputs and outputs. It is then discussed how Apache Hadoop differs from the prototype implemented. The implementation walks through each step of the development process explaining how the prototype has been implemented.

## INTRODUCTION

Big data is large data sets that are too vast and complex that normal data processing is inadequate to handle. With newer programming models, such as MapReduce, it has become much easier to analyse these large data sets into useful bits of information. This is important for companies in the modern age as it allows them to constantly adapt their businesses to match real world parameters. For example, recommendations on the next film for a user to watch or how to more effectively sell a product.

MapReduce is a programming model designed to analyse big data sets, generating useful readable outputs. Map reduce does this efficiently by using parallel and distributed algorithm on a cluster. A widely used implementation of MapReduce is the Apache Hadoop framework. Hadoop implements a distributed file system, called the Hadoop Distributed File System (HDFS), which allows them to break up the input data into large blocks and distribute them across clusters.

The report will discuss how a non-MapReduce prototype was designed and implemented for the task objectives. The objectives were:

- Determine the amount of flights departing from each airport; including a list of airports not used
- Create a list of flights based on Flight ID, outputting the passenger ID, relevant airport codes, departure time, arrival time (in HH:MM:SS format) and flight duration.
- Calculate the number of flights from each airport

## DESIGN

### A detailed description of the MapReduce functions you are replicating

Hadoop's implementation of MapReduce, being the most widely used, was used as a reference for this prototype programme.

**Mapper**
The mapper filters and sorts each row, passed to the mapper, by a certain value, called the key. The output of the mapper is a list of key-value pairs. The key is the value the mapper groups values by and the value is the relevant filtered data. An example of this could be a key of studentID and a value with the rest of the data on that student.

Hadoop's implementation of the `*map()*` function splits the input data rows and runs the mapper function on individual rows which are all executed and filtered in parallel. In this prototype the mapper has an input of chucks of data containing more than one row. This means that each mapper will handle multiple rows outputting a list of key-value pairs. Although, with a more powerful computer this could be changed to split the data rows into smaller blocks, down to one row per mapper. Both Hadoop and the prototype have multithreaded mappers. However, the prototype runs on a limited amount of threads based on how many chunks there are, which are split up into equal chunks of 10 rows.

**Shuffle & Sort**
The shuffle and sort methods takes the output, key-value pairs, of the mapper and prepares it for the reducer. The shuffler retrieves all the mapper outputs and collates them together. The sort then collects all of the same keys from the shuffler grouping all the values together. This makes a key-list(values), so for each key there is a list of associated values.

Hadoop includes the shuffler in the first phase of the reducer. It gathers all the mapper outputs over the HTTP network due to the distributed nodes running the mappers [1]. The Sort is then the second stage, in which it sorts the shuffled values by their keys. These are run simultaneously, meaning as soon as they are fetched they are then sorted. The prototype runs the shuffler and sort separate to the reducer function as one function called '*shuffler()*', using a hash map to store keys and the associated list of values. This is to abstract some of the complexity of the reducer allowing for ease of use for threading the reducer.

**Reducer**
The reducer method generates the output for each of the keys outputted from the mapper. The reducer takes in a key-list(values) and outputs a useful summary for that key. For example, if the input to a reducer was the studentID key-value pair, the reducer could summarise this information by outputting how many classes were attended by that student.

The `*reducer()*` is the third phase of Apache Hadoops Reduce. The method is called for each key in from the shuffle and sort methods. The output is then defined and returned with a reference to the key. The key is used to know which key the output was for as it's run on distributed nodes. The prototype implements the reducer in a similar way with the input being a key-list(values) and then formatting the output for the relevant objective. However, the prototype returns a `*ReducerOutput*` object with reference to the output in two forms, for a text file and csv file. It does not return with a reference to the key as it is not needed when formatting the final output due to the system not being distributed over files or systems.

## IMPLEMENTATION

### The high-level description of the development of the prototype software

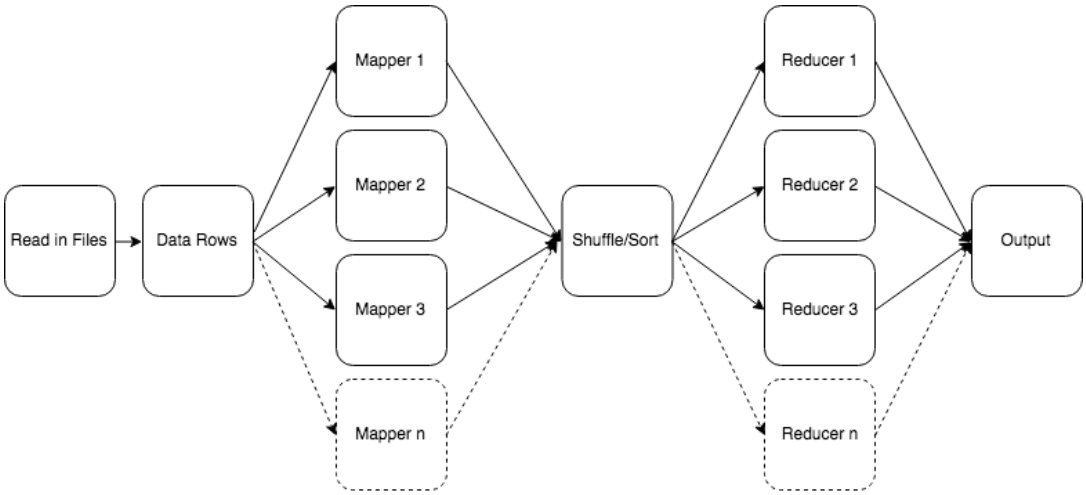


*Figure 1 - Flow of data for non-MapReduce*

The first task in the development was to make functions to read in both the passenger data and airport data files. The was achieved using a BufferedReader to read in the csv's line by line, storing each line in an array for each file. This made it easier to pass to the mapper later on.

'*Airport'* and '*PassengerFlight'* classes were then made. This was to easily store all the relevant data for each row in the respective files.

After both files are read in the airport data is input into a Hash Map. This allows the ease of search for a particular airport and their relevant values, stored as an instance of the '*Airport'* class.

Mapper two was then implemented as a stand alone function that was able to take an input of an array of strings, being the rows of data. Mapper two was implemented first as it was clear that the same mapper could be used for both objective two and three. The mapper loops through all the rows passed to it and splits each row into the different columns. The columns are then passed to the '*PassengerFlight'* class to insatiate and check for errors, which is explained in the section below. If there are no errors then the '*flightId'* and the '*PassengerFlight*' are used to instantiate the '*MapperOutput*' class which acts as a key-value pair. This key-value pair is then added to an array which is returned after all rows of data have been executed.

Mapper one was then implemented as another stand alone function, copying the main body of mapper two, but changing the key and value passed back as the key-value pair. The key was changed to the source airport (which airport the flight was departing from) and the value was the flight ID. This is changed for mapper one as the objective is based on each airport counting each flight, instead of it being based of the flight.

Next the shuffler was made which included both the shuffling and sorting phases of the reduce job in Apache MapReduce. First an array of all the mapper outputs was made and passed it to the shuffler function. The shuffler function goes through each mapper output and adds the key and value to a hash map, with the value being added to a new empty array. If the

key is already entered into the hash map then the value is added to the array of the same key in the hash map. This creates a key-list(values) output, matching the sort phase.

Three reducer functions were then made inputting a key and an array of values. This so that the reducer functions can be run for each key in the hash map from the shuffler. The reducers create two output strings with the relevant data, both in text format and CSV format to be output to different file types. These are returned in instance of the '*ReducerOutput*' class returning both values to where they are called.

After all reducers are called the results are combined and output to console as well as the relevant output strings being made into both '.csv' and '*.txt*' files.

After these were successfully ran serially, outputting the correct files multithreading was then implemented for both the mapper, by dividing the rows into chunks, and the reducer, running for each key in the shuffler has map. This was done in the one class '*ThreadClass'*. The class has two constructors via each type of function. For the mapper it takes in the rows to map and the which mapper to execute. For the reducer it takes in the key it is executing, the associated values and which reducer to run. It is multithreaded using the Java '*Runnable*' implementation. This allows each thread to be joined, using '*.join()*' so that it will wait for all threads to finish once executing.

Figure 1 shows a diagram for the flow of data for one objective. The same flow is used for each objective with their relevant mapper and reducer.

Once the MapReduce flow was fully working a simple GUI was made in which the user can enter the file path for both the input and output files and decide what type of output files should be generated. This can be seen in figure 2.

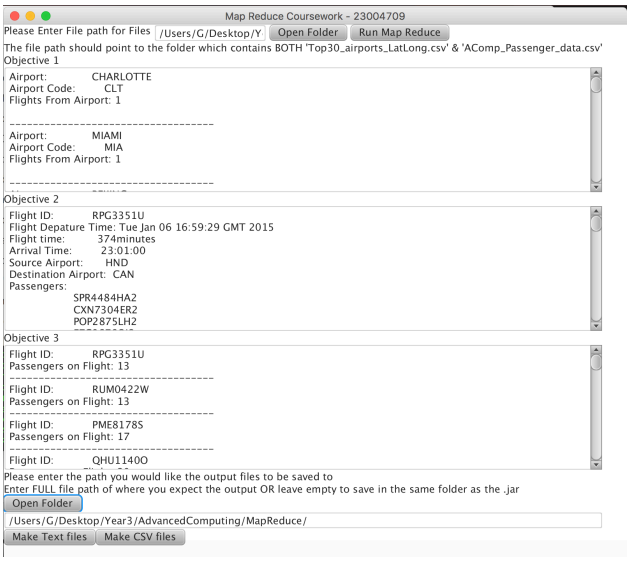


*Figure 2 - Graphical User Interface showing the results of the MapReduce objectives*

**The strategy derived to handle input data error detection/correction and/or run-time recovery**

All errors are output to a '*.log*' file in the same directory as the '*.jar*' when it is run.

Error detection for the passenger data is mostly done in the mapper function. This is so that any row with an error will not influence the final output as well as having the chance to be corrected. There are three different types of errors. The first type of error is syntax checking for each passenger data field. An example of this is the airport code being 'XXX', where 'X' is a capital letter. This error detection is carried out on the instantiation of a '*PassengerFlight*' object. If there is an error with the syntax the '*error'* property will be true so that the '*errorMessage*' property can be added to the list of errors that is output to file. The second error reports if any values have not been filled out with the comment '*Values missing'* returned to file. The last error is when, either the source or destination, airport has a flight code that is not listed in the airport data csv. This is done by checking the airport hash map created, as mentioned in the above section.

There is error correction for the passenger ID as in some cases there is a unreadable character such as '*\uFEFF*' that appears at the start of the character. The error correction will then remove that character and check again that the rest of the passenger ID is valid. If there is no other errors then the error correction will be message will be output to the error file.

**The output format of any reports that each job produces**

The results of each job are output into the relevant boxes for each objective in the GUI shown above in figure 2.

For each MapReduce job both a '*.txt'* (text) and '*.csv*' (Comma Separated Values) files. These are saved as '*ObjectiveX.txt*' or '*ObjectiveX.csv*', where '*X*' is the number of the objective. Sample of the outputs for the objectives as well as screenshots of the error files can be found below.

**Objective 1**

```
Airport:              CHARLOTTE
Airport Code:         CLT
Flights From Airport: 1

-----------------------------------
Airport:              MIAMI
Airport Code:         MIA
Flights From Airport: 1

-----------------------------------
Airport:              BEIJING
Airport Code:         PEK
Flights From Airport: 1

-----------------------------------
```

```
Missing Airports:
          LAX,LOS ANGELES
          IST,ISTANBUL
          DXB,DUBAI
          FRA,FRANKFURT
          SFO,SAN FRANCISCO
          PHX,PHOENIX
          HKG,HONG KONG
          SIN,SINGAPORE
```

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Airport | Airport Code | Flights From Airport | |
| 2 | CHARLOTTE | CLT | 1 | |
| 3 | MIAMI | MIA | 1 | |
| 4 | BEIJING | PEK | 1 | |
| 5 | MADRID | MAD | 1 | |
| 6 | BANGKOK | BKK | 1 | |
| 7 | MUNICH | MUC | 1 | |
| 8 | AMSTERDAM | AMS | 1 | |
| 9 | NEW YORK | JFK | 1 | |
| 10 | LONDON | LHR | 1 | |

| Airport Code | Airport Name |
|---|---|
| LAX | LOS ANGELES |
| IST | ISTANBUL |
| DXB | DUBAI |
| FRA | FRANKFURT |
| SFO | SAN FRANCISCO |
| PHX | PHOENIX |
| HKG | HONG KONG |
| SIN | SINGAPORE |

*Figure 3 – Outputs of '.txt' (left) and '.csv' (right) for objective 1*

## Objective 2

```
Flight ID:            RPG3351U
Flight Depature Time: Tue Jan 06 16:59:29 GMT 2015
Flight time:          374minutes
Arrival Time:         23:13:29
Source Airport:       HND
Destination Airport:  CAN
Passengers:
                      SPR4484HA2
                      CXN7304ER2
                      P0P2875LH2
                      EZC9678QI2
                      PAJ3974RK1
                      JBE2302VO1
                      WBE6935NU1
                      VZY2993ME2
                      ONL0812DH1
                      SJD8775RZ2
                      HCA3158QA1

-----------------------------------
```

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | Flight ID | Flight Depatı | Flight time | Arrival Time | Source Airpo | Destination / | Passengers | |
| 2 | RPG3351U | Tue Jan 06 1( | 374 | 23:13:29 | HND | CAN | SPR4484HA2;CXN7304ER2; | |
| 3 | RUM0422W | Tue Jan 06 1( | 194 | 20:12:59 | MUC | MAD | PUD8209OG0;HGO4350KK( | |
| 4 | PME8178S | Tue Jan 06 1' | 1322 | 15:15:29 | DEN | PEK | CYJ0225CH4;CKZ3132BR8;K | |
| 5 | QHU1140O | Tue Jan 06 1' | 1133 | 12:07:58 | CDG | LAS | BWI0520BG6;MXU9187YC7 | |
| 6 | JVY9791G | Tue Jan 06 1' | 1189 | 13:05:01 | PVG | FCO | WYU2010YH7;CDC0302NN( | |
| 7 | FYL5866L | Tue Jan 06 1' | 1751 | 22:36:40 | ATL | HKG | CKZ3133BR3;SPR4485HA2; | |
| 8 | YZO4444S | Tue Jan 06 1' | 2027 | 03:15:50 | BKK | MIA | UES9152GS6;CYJ0225CH9;( | |

*Figure 4 - Outputs of '.txt' (left) and '.csv' (right) for objective 2*

## Objective 3

```
Flight ID:            RPG3351U
Passengers on Flight: 11
-----------------------------------
Flight ID:            RUM0422W
Passengers on Flight: 9
-----------------------------------
Flight ID:            PME8178S
Passengers on Flight: 14
-----------------------------------
Flight ID:            QHU1140O
Passengers on Flight: 13
-----------------------------------
Flight ID:            JVY9791G
Passengers on Flight: 17
-----------------------------------
```

| | A | B | C |
|---|---|---|---|
| 1 | Flight ID | Passengers on Flight | |
| 2 | RPG3351U | 11 | |
| 3 | RUM0422W | 9 | |
| 4 | PME8178S | 14 | |
| 5 | QHU1140O | 13 | |
| 6 | JVY9791G | 17 | |
| 7 | FYL5866L | 14 | |
| 8 | YZO4444S | 15 | |
| 9 | GMO5938W | 17 | |
| 10 | SQU6245R | 16 | |

*Figure 5- Outputs of '.txt' (left) and '.csv' (right) for objective 3*

## Errors

```
Error Correction at: 1: Unreadable character detected and Corrected UES9151GS7
Error at 6: Error: Syntax Error with Passenger Flight;, FlightID
Error at 9: Values missing
Error at 26: Error: Syntax Error with Passenger Flight;, PassengerID
Error at 41: Error: Syntax Error with Passenger Flight;, FlightID
Error at 47: Error: Syntax Error with Passenger Flight;, FlightID
Error at 53: Values missing
Error at 55: Starting airport does not exist in airport list (UGK)
Error at 59: Error: Syntax Error with Passenger Flight;, FlightID
Error at 95: Error: Syntax Error with Passenger Flight;, PassengerID
Error at 126: Destination airport does not exist in airport list ([AS)
```

*Figure 6 - '.log' file showing errors for the run of objective 1*

## A simple description of the Subversion command line process undertaken

Git was used throughout the development, as a sub versioning tool. Git is a free open source distributed version control system [2]. Using git locally allows you to have a local subversion of the project. This allows you to checkout to different branches of code, to work on different features, or rollback on your commits allowing you to retrieve previous code. Commits includes and changes made to the project. Git can be used with a remote origin as well so that it can be pushed to a server to store you files. A well known Git server is GitHub in which you can push your code for free to store your projects. GitHub was used while developing the project.
When you want to commit changes there are four commands used for the command line interface:

- `*git status*` - Check which files are staged or unstaged (included or not included in the commit)
- `*git add .*` - Adds all unstaged files to the commit
- `*git commit –m "commit message goes here"*` - *Commits the changes to your local repository with a commit message*

- *`git push`* - Pushes all commits that have not yet been pushed to the remote git repository

To see the code on GitHub go to https://github.com/jagribble/MapReduce

## SELF-APPRAISAL

The prototype program that was developed for the assignment met all three objectives, while successfully following the MapReduce model which is highlighted in figure 1 and on Apache Hadoop website [3]. However, there are ways in which the program could be improved.

One way in which the programme could be would be to add more error detection. This would mainly be on with the airport data as currently there is no error detection on the data, as from the file received there were no invalid airports apart from empty rows which is already handled. This would allow for new data to be added the list while not introducing errors to the outputs. Another improved error detection could be checking if each passenger isn't on two flights at the same time. This could be achieved by chaining MapReduce jobs.

Another enhancement to the program would be to implement a distributed system such as the one implemented by Apache Hadoop. Although, it may be unnecessary for the small data set provided for the assignment, implementing a distributed system would be able to scale well to a much bigger dataset.

## REFERENCES

[1] Apache Software Foundation, '*Reducer'* [Online.]. Available: https://hadoop.apache.org/docs/r2.7.0/api/org/apache/hadoop/mapreduce/Reducer.html

[2] Git [Online.]. Available: https://git-scm.com/

[3] Apache Software Foundation, '*MapReduce Tutorial'* [Online.]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html