# Develop a Software Prototype of a MapReduce-like system

## Introduction

This report aims to detail the steps that were taken in order to implement a software prototype of a system that mimics the function of MapReduce. The MapReduce algorithm consists of two procedures, which are called Map and Reduce. Initially, the Map function takes in input data and coverts it into a new form of data, where individual elements are broken down into tuples (key/value pairs). Following, the Reduce function takes the output produced by the Map and combines the key/value pairs into a smaller set of key/value pairs in order to reduce the set of data[1].

MapReduce provides the ability to easily scale data processing. This is due to its ability to store as well as distribute large data sets across many servers. Considering these servers can operate in parallel, the addition of extra servers simply increases performance. By dividing tasks in a manner that allows their execution in parallel, multiple processors can take on these divided tasks and this allows the execution of programs to occur very quickly[2].

## High-level Development

In order to begin development, the structure of a Hadoop job was firstly considered in depth[3]. Each MapReduce process is split into a process named a job. Each job will firstly set the input and output data location. Afterwards, the job will set the mapper class to the user supplied mapping code and the reducer class to the user supplied reducing code. Once this has been completed, the job will execute as demonstrated in the diagram below in Figure 1.

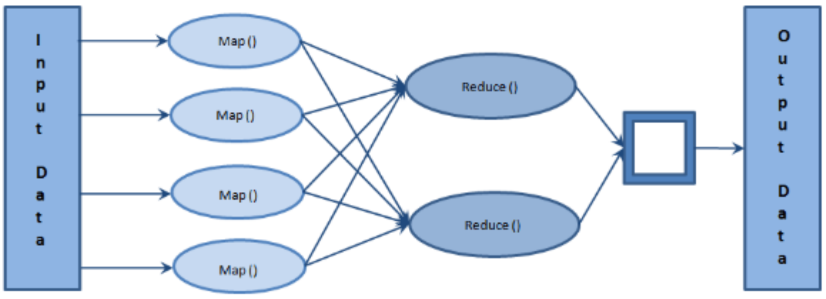

Figure 1 - Map Reduce Model

When the job executes, the mapping tasks will run with the supplied input data. Each mapping task will execute in parallel. Once all the mapping tasks have been completed, the reduce tasks will follow in parallel execution. Lastly, after both the map and reduce tasks have been completed, the output data will be written to a file[4].

## Git command line process

Version control is important for documents that undergo a lot of revision and redrafting and is particularly useful for maintain a backup of source code[5]. For this project, the version control system utilised was GitLab. The project can be found at the following link:

https://csgitlab.reading.ac.uk/cx019583/AdvancedComputing_MapReduce

## MapReduce replicated functions

Each time a MapReduce job is executed, there will be consistent elements that will be reused. The classes that will not be reused every time are the Main class, the Map class and the Reduce class. The Main class will determine the number of jobs and instantiate them whereas the Map and Reduce classes will require the user to supply code depending on the each individual task. As the Map class will always be required to have a map function, an abstract class named Mapper was created to provide this function and allow for code reusability. The Mapper also implements the type Callable so it can be called from the job utilising multithreading. Similarly, the Reduce class will always be required to have a reduce function, therefore another abstract class was created named Reducer in order to provide said function. The Reducer class also implements the type Callable. When the map function executes, it converts input data into a key/value pair, therefore it was required to create a kay/value pair object in order to store the result of the mapping. Consequently, the Reducer takes this key/value pair as input and this code is reused across the system. The last class created was Job, which provides functions for setting the input and output data as well as setting the Mapper and Reducer classes each time. Job provides the private functions map, reduce, finish, which are executed in order when the public function run is called.

## Output

Objective 1 - Determine the number of flights from each airport; include a list of any airports not used:

```
ORD,2
PVG,1
LAX,0
CDG,1
CLT,1
JFK,1
DFW,1
LHR,1
MUC,1
SFO,0
PHX,0
KUL,2
BKK,1
MIA,1
HKG,0
CGK,2
AMS,1
IST,0
DEN,4
DXB,0
MAD,1
CAN,2
IAH,2
FCO,1
FRA,0
PEK,1
UGK,1
ATL,2
SIN,0
HND,1
LAS,1
```

Objective 2 - Create a list of flights based on the Flight id, this output should include the passenger Id, relevant IATA/FAA codes, the departure time, the arrival time (times to be converted to HH:MM:SS format), and the flight times:

```
QHU1140O, from CDG to LAS
Departed: 17:14:58, 06-01-2015 (GMT)
Arrived: 12:07:58, 07-01-2015 (GMT)
Duration: 18 hours and 53 minutes
Passengers:
        MXU9187YC7
        MXU9187YC7
        SJD8775RZ8
        HCA3159QA0
        CDC0302NN5
        BWI0520BG6
        LLZ3798PE5
        LLZ3798PE5
        WBE6935NU6
        PUD8209OG7
        BWI0520BG6
        SJD8775RZ8
        UES9151GS8
        PAJ3974RK5
        CKZ3133BR0
        JJM4724RF9
        LLZ3798PE5
        JBE2302VO6
        WBE6935NU6
        CKZ3133BR0
```

Objective 3 - Calculate the number of passengers on each flight:

```
QHU1140O,21
FYL5866L,20
BER7172M,17
RPG3351U,13
KJR6646J,23
HUR0974O,7
ATT7791R,15
VYW5940P,17
HZT2506M,14
VDC9164W,14
EWH6301Y,10
WSK1289Z,21
TMV7633W,15
PME8178S,17
XOY7948U,16
RUM0422W,14
MBA8071P,15
XXQ4064B,24
GMO5938W,24
SOH3431A,18
SQU6245R,20
VYU9214I,15
XIL3623J,13
JVY9791G,20
MOO1786A,13
WPW9201U,11
DAU2617A,12
ULZ8130D,27
YZO4444S,17
PNE8178S,1
DKZ3042O,11
```

Objective 4 - Calculate the line-of-sight (nautical) miles for each flight and the total travelled by each passenger and thus output the passenger having earned the highest air miles:

```
CYJ0225CH1,2385
CYJ0225CH0,1161
CYJ0225CH3,4407
CYJ0225CH2,3344
CYJ0225CH5,5425
CYJ0225CH4,5509
CYJ0225CH7,5827
CYJ0225CH6,4371
CYJ0225CH9,8445
CYJ0225CH8,7546
YMH6360YP4,4407
DAZ3030XA1,7015
BWI0521BG0,7702
YMH6360YP5,4174
YMH6360YP2,3344
```

## Error Handling Strategy

The error handling strategy that was implemented is to use regular expression pattern matching on the data in the map function[6]. This is handled by a class named Validate which provides regular expressions for each data variable. If the pattern matches the regular expression in Validate then the Mapper will convert the data to a key value pair. If the data doesn't match, a line will be skipped and an error message will be presented to the user in the console. This assists to account for missing value data as well as incorrectly typed data.

## Conclusion

Given the above, it can be concluded that the requirements of the specification were met since the software which was developed replicates the MapReduce system successfully. The two main procedures of the algorithm were implemented through the Mapper and Reducer classes and the error handling was achieved through the use of the class Validate.

The project could be further improved upon by reconsidering the way error handling was implemented. This could be changed so that when the input data is set, it is firstly parsed to remove any potential errors in the data before forwarding the data to the mapper. Furthermore, there could be improvements regarding the overall system design by implementing a distributed client-server infrastructure.

Through the development of this MapReduce-like system, a better understanding of the algorithm was acquired.

## References

[1] https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm
[2] https://www.quora.com/What-are-the-advantages-of-MapReduce
[3] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
[4] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[5] https://www2.le.ac.uk/services/research-data/organise-data/version-control
[6] https://zeroturnaround.com/rebellabs/java-regular-expressions-cheat-sheet/