CS3AC16

Advanced Computing

Develop a Software Prototype of a MapReduce-like System

Student ID: 22002003

Abstract

This report details the design, implementation and and development of a software system that replicates Map and Reduce functions from the MapReduce programming model. The system allows for the parallel execution of data processing tasks and therefore provides the building blocks to support a framework which mimics the execution of Hadoop without a client-server infrastructure.

Introduction

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster[1]. MapReduce is the heart of Apache Hadoop and the term "MapReduce" refers to two separate and distinct tasks that Hadoop performs. The first of these tasks is the "Map" job, in which the mapper takes a set of data and breaks down the individual elements into tuples of key-value pairs. Following the execution of the "Map" job, the "Reduce" job take the output from the "Map" job as input and combines the key value pairs into a smaller set, therefore reducing data[2].

The essence of a MapReduce system is that the system is able to process data (map, then reduce) in parallel. The purpose of parallelism is to provide scalability and cost-efficiency in processing large data sets, thus ensuring high performance.

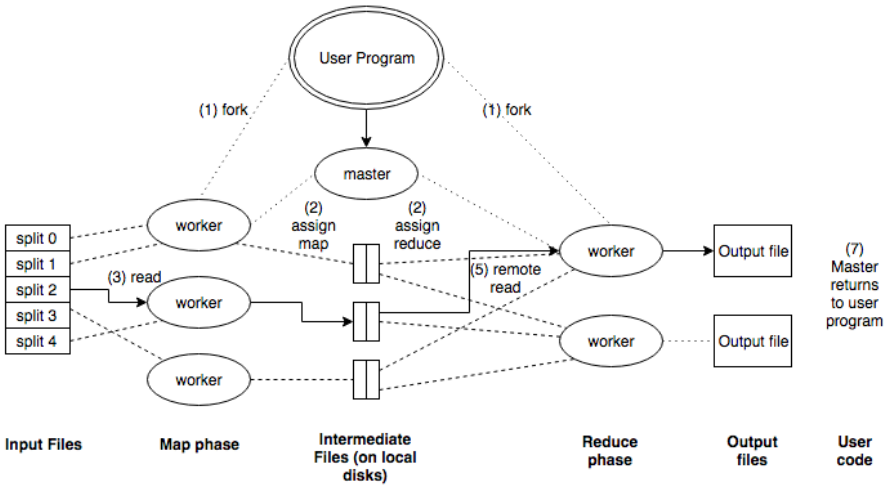

Figure 1.
MapReduce Workflow

In Figure 1 above, the workflow of the MapReduce algorithm can be visualised. In the first step, the master divides the input data into equal parts and provides this as input to each worker. In the second step, there are M map tasks to assign and R reduce tasks to assign. Depending on the number of threads available, the master assigns the tasks from the threads available in the pool to the workers. In the third step, a worker assigned a map task reads in the provided input. By executing the map function, the input data is converted into key-value pairs which are buffered in memory. In step four, the buffered key-value pairs are written to the local disk. The location of the map results are then passed back to the master, who forwards these locations to more workers to reduce. In the fifth step, when a reduce worker gets the location from the master it reads the buffered data from the disks. Once all the intermediate data has been read, the keys are sorted in order to group all of the same occurrences together. In the sixth step, the reduce worker iterates over the sorted intermediate data and passes the key and corresponding values to the user's Reduce function for each unique intermediate key encountered. The output of the reduce function is then appended to a final output

file. In the seventh and final step, once all map and reduce tasks have been executed, the program returns back to the user code[3].

High-Level Development

In order to develop a map-reduce like system, the architecture of Hadoop was firstly considered. Hadoop consists of three core components which are as follows:

- Hadoop Distributed File System (HDFS)
- MapReduce
- Yet Another Resource Negotiator (YARN)

The HDFS is the file system of a Hadoop cluster. HDFS is slightly different to other distributed file systems, considering it is designed with hardwire failure in mind, it is optimised for sequential operations and is cross-platform and support heterogenous clusters. Data in a Hadoop cluster is broken into blocks and distributed throughout the cluster. Each block is duplicated twice, with two replicas stored on two nodes in a rack elsewhere in the cluster. Since the data has a default replication of three, this ensures the system to be highly available and fault-tolerant.

In the MapReduce algorithm, the top-level unit of work that is executed is a job. Each job is composed of one or more map or reduce tasks. When considering a MapReduce job of counting the frequency of words in text, figure 2 below illustrates the core execution of operations.

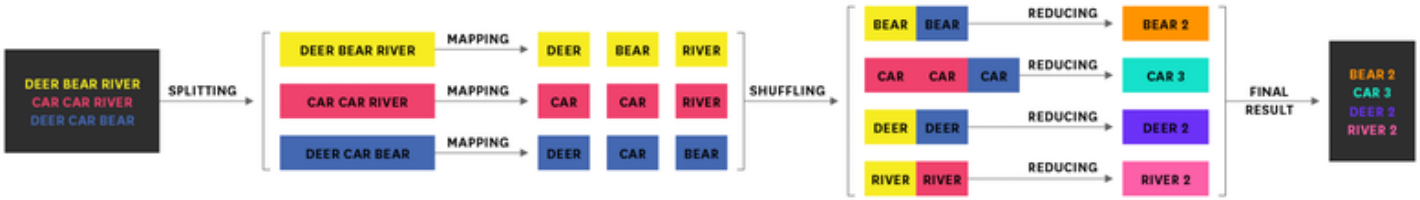

Figure 2.
MapReduce Word Count Problem

Hadoop also utilises YARN in order to efficiently assign computation resources for application execution. Yarn also consists of three components - ResourceManager, ApplicationMaster and NodeManagers. YARN is not only used to allocate resources in Hadoop, considering in its latest versions YARN is combined with ZooKeeper in order remove any single points of failure from the application[4].

In order to implement a software prototype of the MapReduce algorithm, the implementation utilised is a simple form of the implementation used in Hadoop. Therefore, the only component from Hadoop being replicated will be the MapReduce system and implementing a Distributed File System in combination with Yarn will not be considered. The software prototype of a MapReduce systems works from the principle that each MapReduce workflow operates as a job. When instantiated, a job will set the map class to the user supplied code, along with setting the reduce class to the relevant user supplied code. The job will then set both the input and the output files and once this is completed, this job will execute.

The execution process will firstly assign mapping tasks to each available thread and the key-value pairs created from the input will be stored. Following the execution of the mapping task, the reduce tasks will be assigned to threads and then executed, receiving the output of the mapping task as input and reducing the key-value pairs. Finally the job will write the write the resulting data to a file and upon completion the job will have finished.

Subversion Control

The subversion command line process undertaken utilised GitLab. The repository is available at the following location: https://csgitlab.reading.ac.uk/mm002003/MapReduce

When developing the project, initially a version of the word count problem was created in order to ensure that the MapReduce system was working as intended. Following the creation of a tested MapReduce system, the tasks were implemented in order to solve the problems provided. Subversion control in an important aspect of software development, as it allows developers to maintain current and historical versions of files[5].

Software Prototyping

In order to begin prototyping the system, software development best practices must firstly be considered. As the project will be developed in Java, this includes utilising an object-oriented paradigm. Furthermore, the software should be developed in a way in order to provide as much code re-usability as possible, which can be achieved through providing a clear modular structure for the programs and utilising inheritance[6].

When considering a MapReduce job, there are some code elements that will change with every iteration depending on the task, such as the Map class, the Reduce class and the Main. The Main class is required to instantiate the relevant job(s) and the Map and Reduce classes will require the user to supply code in order to alter the execution of operations. However there will be consistent building blocks which are required each time a MapReduce job is executed. These include the classes Job, KeyValuePair, Mapper and Reducer.

The class KeyValuePair is created in order to store the results of the Map operation. This object is also required as an input to the reducer as well as an output.

The class Mapper is an abstract class created, which defines the abstract function map. Each Map class that the user writes will extend this Mapper class, therefore inheriting its variables and functions required. The Mapper class also implements Callable in order to provide an interface for multithreading through the job.

The class Reducer is an abstract class created, which defines the abstract function reduce. Each Reduce class that the user supplies code for will extend the Reducer class, providing the function of inheritance and therefore code-reusability. The Reducer class also implements Callable in order to provide an interface for multithreading.

The job class is created to control the execution of a MapReduce operation. This is achieved through providing functions to set the input location, set the output location, set the mapper class provided and also set the reducer class provided by the user. The job class has the private functions map, reduce and end, as well as the public function run, which calls each private function in succession. When the required inputs are supplied and the job is executed, the map function will construct a new mapper to map each line of input. This is achieved by creating a thread pool that reuses a fixed number of threads operating off a shared unbounded queue. At any point, the available processors at runtime will determine the threads that will be active processing tasks. This is replicated in the Reduce function, in order to provide a parallel processing system that achieves both map and reduce functions.

Job Output

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

Flights from each airport:

PVG,1
ORD,2
CDG,1
CLT,1
JFK,1
DFW,1
LHR,1
MUC,1
BKK,1
KUL,2
MIA,1
CGK,2
AMS,1
DEN,4
CAN,2
MAD,1
IAH,2
FCO,1
PEK,1
UGK,1
ATL,2
HND,1
LAS,1

Airports not used:

LAX,0
SFO,0
PHX,0
HKG,0
IST,0
DXB,0
FRA,0
SIN,0

Task 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:
- BWI0520BG6
- WBE6935NU6
- CDC0302NN5
- SJD8775RZ8
- WBE6935NU6
- MXU9187YC7
- HCA3159QA0
- UES9151GS8
- MXU9187YC7
- LLZ3798PE5
- LLZ3798PE5
- LLZ3798PE5
- SJD8775RZ8
- JJM4724RF9

JBE2302VO6

Task 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
VDC9164W,14
HZT2506M,14
EWH6301Y,10
TMV7633W,15
WSK1289Z,21
XOY7948U,16
PME8178S,17
RUM0422W,14
MBA8071P,15
XXQ4064B,24
GMO5938W,24
SOH3431A,18
SQU6245R,20
XIL3623J,13
VYU9214I,15
JVY9791G,20
4QU6245R,1
MOO1786A,13
WPW9201U,11
DAU2617A,12
ULZ8130D,27
YZO4444S,17
VDCP164W,1
PNE8178S,1
DKZ3042O,11

Task 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.

Highest Air Miles: SPR4485HA5 - 15714

CYJ0225CH1,2385
CYJ0225CH0,1161
CYJ0225CH3,4407
CYJ0225CH2,3344
CYJ0225CH5,5425
CYJ0225CH4,5509
CYJ0225CH7,5827
CYJ0225CH6,4371
CYJ0225CH9,8445
CYJ0225CH8,7546
YMH6360YP4,4407
YMH6360YP5,4174
BWI0521BG0,7702
DAZ3030XA1,7015
DAZ3030XA2,7015
YMH6360YP2,3344
…

Error Handling/Correction

In order to handle errors that may be present in the dataset, regular expression matching is utilised in the Map class in order to confirm if the data provided matches the expected pattern. If the pattern does not match the provided regular expression, the key value pair is not stored and the process continues. Conversely, if the pattern matches the regular expression and provides correct data, the key value pair is stored. Not only does this ensure that correctly pattern matched data is stored, but by utilising regular expression matching, missing values are appropriately handled and removed from the input.

Whilst this initially provides some form of error handling, there may also be data inconsistency between the files. Therefore data between files provided are cross-referenced in order to ensure that the

results are more accurate and consistent. For example, in the first task the data is combined from both the A_Comp_Passenger_Data.csv file as well as considering the airports from the Top30_airports_LatLong.csv file. This ensures that all of the data is consistent and cross-referenced.

Conclusion

In conclusion, the system developed effectively provides the building blocks to mimic the functions of the MapReduce/Hadoop framework. By constructing the project utilising object-oriented programming, abstract classes and inheritance were incorporated in order to ensure code re-usability throughout the project. The error handling and correction implemented is currently only considered in a limited fashion, mainly focusing on pattern matching of regular expressions. Error handling could be further improved upon by pre-processing the data in order to consider the presence of duplicates, missing values and incorrect data and therefore this would provide better separation of concerns and a more elegant solution for handling errors in the data.

The project could also be further improved upon by altering the way in which the data is allocated to each mapper. Currently, a new mapper is constructed for each line of input from the data, however this can be improved upon by more appropriately allocating resources. For example, when initiating a new job, the input data could firstly be divided into equal portions and then provided to each mapper as a data set. This would therefore further increase the performance of the system.

Furthermore the project could also be improved upon by considering the reliability of the system. Hadoop achieves reliability by removing any single point of failure from the system, which was achieved with the introduction of High availability in Hadoop 2.0. A single point of failure is a risk posed by a flaw in the design of a system in which one fault causes the entire system to stop operating. In order to achieve this, a much more robust system design would be required to be implemented. Throughout the development process I achieved much greater understanding of how the MapReduce algorithm operates and was able to effectively implement this in order to complete the tasks provided.

References

[1] - CNET, Google spotlights data centre inner workings, 2008 [Online]. Available at: https://www.cnet.com/10784_3-9955184-7.html
[2] - IBM, Apache MapReduce, 2018 [Online]. Available at: https://www.ibm.com/analytics/hadoop/mapreduce
[3] - K, Comalli, A MapReduce Overview, 2017 [Online]. Available at: https://towardsdatascience.com/a-mapreduce-overview-6f2d64d8d0e6
[4] - E, Mouzakitis, Hadoop Architecture Overview, 2016 [Online]. Available at: https://www.datadoghq.com/blog/hadoop-architecture-overview/#from-theory-to-practice
[5] - W, Nagel, Subversion Version Control: Using the Subversion Version Control System in Development Projects, 2005 [Online]. Available at: https://dl.acm.org/citation.cfm?id=1036703
[6] - Anon, OOPS Concepts in Java with Examples, 2019 [Online[. Available at: https://www.guru99.com/java-oops-concept.html