Data Processing using MapReduce on the Cloud
James Garbutt
Computing
2011/2012

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)__________________________
Summary

This project aims to develop a good understanding of MapReduce on a cloud, using large amounts of data. The project will first concentrate on implementing a working MapReduce implementation and applications to be executed on it. Afterwards, the performance of these applications will be investigated and evaluated.

This report consists of background research on the related topics, technologies and solutions. After this background research, the development and implementation of both the systems and the applications are to be documented. All of this is to be concluded by an evaluation of the performance and the tools used.
Acknowledgements

I’d like to thank my supervisor, Karim, for taking a good interest in my project and communicating well by scheduling regular meetings. I would also like to thank Django Armstrong for providing good help in regards to the cloud, even in his spare time.
## Contents

1 Introduction 1
   1.1 Project Aim .......................................................... 1
   1.2 Objectives ............................................................ 1
   1.3 Minimum Requirements .............................................. 1
   1.4 Methodology .......................................................... 2
   1.5 Schedule .............................................................. 2
   1.6 Summary .............................................................. 2

2 Background Research 4
   2.1 Introduction .......................................................... 4
   2.2 Overview of Clouds .................................................. 4
   2.3 Cloud Models ........................................................ 5
      2.3.1 Software as a Service (SaaS) .................................. 5
      2.3.2 Platform as a Service (PaaS) .................................. 5
      2.3.3 Infrastructure as a Service (IaaS) ............................ 5
   2.4 Types of Clouds ...................................................... 6
      2.4.1 Private Clouds ................................................... 6
      2.4.2 Public Clouds .................................................... 6
      2.4.3 Hybrid Clouds ................................................... 6
      2.4.4 Community Cloud ................................................ 6
   2.5 Virtualisation ......................................................... 7
      2.5.1 Full Virtualisation ............................................... 7
      2.5.2 Paravirtualisation ............................................... 7
      2.5.3 Hybrid Virtualisation ........................................... 7
   2.6 Data Management ..................................................... 7
      2.6.1 Issues ............................................................ 7
   2.7 MapReduce ............................................................ 8
      2.7.1 Alternatives ....................................................... 10
      2.7.1.1 Master/Slave ................................................. 10
2.7.1.2 Producer-consumer ........................................... 10
2.7.1.3 Shared Memory ............................................. 10
2.7.2 Uses of MapReduce ............................................. 11
2.8 Hadoop ............................................................... 11
  2.8.1 History ......................................................... 12
  2.8.2 Design .......................................................... 12
2.9 Related Work ....................................................... 13
2.10 Summary .............................................................. 13

3 Design ................................................................. 14
  3.1 Introduction ....................................................... 14
  3.2 Methodology ....................................................... 14
  3.3 Example Problem .................................................. 15
  3.4 Solution ............................................................. 15
  3.4.1 MapReduce Model .............................................. 15
  3.5 Application of MapReduce ........................................ 16
    3.5.1 Map .......................................................... 16
    3.5.2 Reduce ....................................................... 16
    3.5.3 Performance .................................................. 17
  3.6 Summary .............................................................. 17

4 Implementation ....................................................... 18
  4.1 Introduction ....................................................... 18
  4.2 The Cloud ......................................................... 18
    4.2.1 Cloud Testbed ............................................. 18
    4.2.2 Infrastructure .............................................. 20
    4.2.3 Hypervisor .................................................. 20
    4.2.4 Limitations .................................................. 21
  4.3 Data ................................................................. 21
    4.3.1 Introduction ................................................ 21
    4.3.2 Fault/Failure Data ......................................... 21
    4.3.3 Weather Data .............................................. 22
    4.3.4 Conclusion .................................................. 22
  4.4 Virtual Machine Setup .......................................... 22
    4.4.1 Operating System .......................................... 23
  4.5 Hadoop Setup ..................................................... 23
  4.6 Hadoop Distributed Filesystem .................................. 24
  4.7 Data Pre-processing ............................................. 24
  4.8 Measures of Performance ....................................... 25
5.4 Hadoop .................................................. 43
  5.4.1 Introduction ........................................ 43
  5.4.2 Issues ............................................. 43
  5.4.3 Limitations ........................................ 43
5.5 Minimum Requirements .................................. 44
5.6 Methodology ........................................... 44
5.7 Existing Work ........................................... 45
5.8 Further Work ........................................... 45

6 Project Conclusion ........................................ 47
  6.1 Discussion ............................................ 47

Bibliography .................................................. 49

A Personal Reflection ........................................ 51
B Materials Used ............................................. 53
C Ethical Issues .............................................. 54
D Sample of Data ............................................. 55
Chapter 1

Introduction

1.1 Project Aim

The aim of this project is to develop an understanding of how MapReduce can be used to process large amounts of data in a cloud environment. Such an understanding will involve investigation of the performance and behaviour during a typical scenario.

1.2 Objectives

The objectives of the project are to:

- Develop an understanding of the MapReduce model and the benefits it has over alternatives
- Successfully make use of a MapReduce implementation over multiple machines
- Understand how to determine which systems/resources are the most reliable depending on historical fault data
- Implement a MapReduce application using this understanding to automatically analyse data and produce a suitable result
- Compare the performance of the MapReduce model with serial execution on a single machine

1.3 Minimum Requirements

The minimum requirements are:
• To find a large data source suitable for processing
• A MapReduce application which produces a useful result from this data
• To investigate the performance of such an application on a cloud
• A working Hadoop implementation on a cluster of at least three nodes

The possible extensions are:

• Develop a MapReduce application for analysis of another data source such as climate data

1.4 Methodology

To meet the minimum requirements associated with this project, a substantial amount of research into the related background topics must be completed. The related background topics being primarily information on current technologies being used in the industry and the architecture of clouds. In addition to this, at least one application must be designed and developed to be executed on the University Cloud for evaluation.

1.5 Schedule

To keep track of the project and ensure it was completed on time, a schedule was created. Each task within the schedule was essentially mandatory, however, the subtasks within one given parent task could vary slightly from the expected schedule. Parent tasks were created to define the limits of a set of tasks, the absolute maximum amount of time to be spent on them.

Due to the nature of this project and it being heavily evaluation based, both the testing and production of the MapReduce applications overlapped with each other. The reason for this being that testing the applications would more than likely result in bug fixes or improvements being added.

1.6 Summary

The minimum requirements and aim have now been stated clearly, so the next chapter will look at the background of the technologies and topics at hand. Of course, a good understanding of the technologies being used is needed, hence the research in the next chapter.
Figure 1: Schedule Gantt Chart

<table>
<thead>
<tr>
<th>Name</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
<th>Week 13</th>
<th>Week 14</th>
<th>Week 15</th>
<th>Week 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Literature Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clouds (History &amp; Research)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clouds (Relation to Data)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clouds (Case Studies)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtualisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapReduce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapReduce Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud Testbed Familiarisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapReduce Familiarisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collect the Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understand the Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement the MapReduce Solution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Produce the Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test The Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalisation &amp; Hand in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2

Background Research

2.1 Introduction

The aim of this chapter is to look into the background of clouds, MapReduce/Hadoop and the cases in which they have been proven to be useful. Most, if not all of the technologies and concepts discussed here will be used in this project.

2.2 Overview of Clouds

A cloud is a form of a distributed system which involves key features such as transparency, reliability and redundancy. There are many different definitions for a cloud, each of which is equally valid compared to the next. However, a cloud as defined by the NIST (National Institute of Standards and Technology) [20] is as follows:

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. [20]

Clouds generally exist to allow utility computing to be possible, the idea of charging on a usage basis rather than a fixed subscription fee. Such an ability allows businesses to sell their services but only charge a fee for what is used, allowing more flexibility than usual.
2.3 Cloud Models

There are three primary cloud models, in a typical cloud all three are used in a layered fashion. Each model has a unique use but may rely on another model as a base, a platform, you could say.

2.3.1 Software as a Service (SaaS)

Software as a service is essentially the idea of hosting an application as a service to the customers who have purchased it and accessed it via the Internet [10].

The clear benefit of this is that no customer needs an actual copy of the application, instead they access an online version which is under complete control of the provider. This also means the customer never has to deal with software updates, bug fixes and so on, simply due to the fact that the provider now handles it [10].

In terms of benefits for the provider, the most obvious one is the fact that the source code remains unavailable and hidden from the user, meaning there’s almost zero chance of source code disclosure occurring, by accident or purposely. Having such control would prevent competitors reverse engineering the application to create their own and would essentially stop any possibility of piracy.

2.3.2 Platform as a Service (PaaS)

This provides the user with a platform on which they can design, develop, test, deploy and host their own applications using only Internet access. Again, nothing important is stored locally but rather stored on the cloud and made accessible via the Internet.

Within this model, the consumer does not control any of the underlying infrastructure but has complete control over the deployed application(s) and often the system environment too [20].

This layer of the cloud structure may often involve the use of virtualisation, meaning the platform mentioned here could and probably would be a virtual machine, allowing the consumer to alter it in any way they like without affecting the cloud.

2.3.3 Infrastructure as a Service (IaaS)

The lowest layer in a classic cloud structure and the model in which the consumer is given the cloud’s actual hardware as a service. This allows the cloud provider to purchase resources and equipment which are then shared with the consumer to be used for anything they please [10].

This is clearly beneficial due to the ability to expand and acquire more resources (hardware) when required. The alternative, older method being to purchase the equipment, rack space and connectivity yourself. This way, the consumer need not worry about hardware usage or capacity as it can be added to in future without interrupting service.
2.4 Types of Clouds

There are various different types of clouds which are in use today, each of which is useful for a different reason than the others, in most cases. The main types of clouds are hybrid, public, private and community clouds.

2.4.1 Private Clouds

Private clouds are used solely by an organisation rather than being offered to third parties [20]. Though the cloud is private and used only by the organisation which owns it, that does not mean they are the ones operating it. It is common for a private cloud to be managed and operated off-site by a third-party company but used only by the organisation paying for it. One major benefit of having such a cloud is the fact that all data is accessible only by the organisation and is not shared across servers being used by other businesses.

2.4.2 Public Clouds

Public clouds are made available to the general public but remain under the ownership of a single organisation [20]. These are the most common clouds available today as anyone can purchase resources from them for whatever requirements they may have. Usually, public clouds are very well documented to help consumers with development and usage of the provided resources. Additionally, they generally charge the consumer for what is used in terms of bandwidth and processing power rather than fixed fees for a fixed amount of resources.

2.4.3 Hybrid Clouds

Hybrid clouds are a combination of two types of cloud, often private and public clouds but occasionally community clouds may be used too [20]. The use of this is so that if the master cloud were to run out of resources, it could burst and begin making use of a secondary, third-party cloud. Such a concept allows the initial cloud to be small to save on costs, as it will always have resources available due to the ability to burst.

2.4.4 Community Cloud

A community cloud is a cloud shared by several organisations with some common concern/task in mind [20]. Such a cloud could be managed and operated by a third party, on-site or off-site, essentially like a public cloud but used by people with common interests.
2.5 Virtualisation

Virtualisation is used heavily in most clouds to provide a single platform distributed across the cloud. In most cases, the consumer is provided with a virtual machine or multiple virtual machines, allowing them to alter anything they like without affecting the underlying infrastructure.

2.5.1 Full Virtualisation

This is the type of virtualisation in which all virtual machines run within another system, independently and isolated [10]. In this case, all virtual machines run on top of another operating system, which it’s self runs directly on top of the hardware [25]. There is a slight amount of overhead involved with full virtualisation as almost everything is emulated for each virtual machine.

2.5.2 Paravirtualisation

In this environment, the hypervisor runs directly on the hardware so the virtual machines can interact directly with it, rather than it all being emulated [25]. The overhead involved is less than that of full virtualisation due to the hardware absence of a layer between the hypervisor and hardware so is a more common choice.

2.5.3 Hybrid Virtualisation

Hybrid virtualisation involves both full virtualisation and paravirtualisation. The virtual machines are fully virtualised but make use of paravirtualised drivers for hardware components such as storage devices and network interfaces [21]. In this case, because the storage is paravirtualised, disk performance increases overall due to the fact that there is one less layer between the machine and the disk.

2.6 Data Management

One primary reason why clouds exist and are used is for the benefit of data management, the ability to process and manage large amounts of data, often in parallel. Data processing and management plays a large part in almost all projects as they all involve some form of data, whether it be large or small.

2.6.1 Issues

Although the implementation of a certain application involving data management may be simple, certain issues do exist. Fault tolerance, data distribution and parallelisation are just three potential features which must be implemented to prevent low efficiency or faults [13].

If the data is of a large size, it becomes clear that some form of parallelisation and data distribution is required in order to reach a high level of efficiency. Otherwise, the data would be processed in
serial on a single system, meaning a potentially large amount of processing time required. So the clear choice would be to distribute the data across multiple systems and have each system process the chunk it has been given, meaning it will complete in a far lower amount of time.

Another issue is heterogeneity, the idea that data may be stored on systems which differ in terms of hardware and software. This is a common issue in most datacenters as they often own multiple generations of hardware and, in the case of clouds, provide various different types of virtual machine [27].

Both heterogeneity and large data collections are issues this project will encounter, though the latter is most likely. Considering the fact that the entire project will be handled by a collection of almost identical virtual machines, heterogeneity is not a huge problem. However, the amount of data being used is likely going to be of a large size, meaning the distribution and handling of it is important to be taken into account during development.

In relation to data sizes, distribution of the data is another potential issue. Ideally, we would like the data to be distributed evenly across all machines involved, possibly with some form of redundancy to assist with fault tolerance.

2.7 MapReduce

The general idea of the MapReduce model is to process a set of key-value pairs using a Map function provided by the developer. After this function has been executed on each pair, you are then left with a set of unique keys, each of which may have multiple values from before. At this point, a Reduce function is used to merge the values of each key into a potentially smaller set of values, often a single value [13].

The MapReduce model involves several stages which run sequentially as follows [13]:

• The initial input to the user-created program is split into many pieces by the master node
• Each worker is given a copy of the user-created program to store locally for execution in future
• Each idle worker is then assigned a map or reduce task
• Workers assigned a map task take their provided chunk of data and pass each key-value pair to the user-defined Map function
• Map workers then write their results, a collection of intermediate key-value pairs, to disk (often a distributed filesystem)
• Workers assigned a reduce task are given the location of the data they must process. They then read this data, sort it and group it by key, then pass each pair to the Reduce function.
• Reduce workers then write their results to an output file which the master has access to
In addition to these stages, some fault tolerance is also implemented to ensure that execution is successful [13]. The master also pings each worker on a regular basis to check if it is still alive or has failed. Once a worker is marked as failed, any tasks it has been given are then taken back to be rescheduled and each worker is notified of this change. Some workers may also be marked as a "straggler", meaning it is taking an unusual amount of time to process the task it has been given. In this case, the task is given to a secondary worker as backup and the first of the two to finish causes the other to be cancelled.

Although most users only wish to alter the map and reduce functionality, it is possible to make use of custom partitioning and combiner functions. In some cases it would be beneficial to have a custom function used for partitioning the data, usually to ensure that it remains in a specific order or related entries are processed by the same worker. A custom combiner function would allow the user to combine entries, similar to how the reduce function works, before they are passed through the actual reduce function.

The MapReduce model is most commonly paired with some form of distributed file system. The reason for this being that if all workers and the master share the same distributed file system, they may read and write data to the same source without the need to communicate with specific nodes. So when the map workers store their output, they will store it in this file system which a reduce worker may then access directly without worrying about handling the underlying communication and data transfers. The same applies for the reduce workers, they are able to read the input directly from the file system and store their output in the same location for the master to use.
2.7.1 Alternatives

Although Hadoop has become the standard framework to use for data processing, certain alternatives do exist, often with specialist uses and features. For this project specifically, we will not be using an alternative implementation as Hadoop meets all our requirements while also being very efficient.

2.7.1.1 Master/Slave

One such alternative to the MapReduce model entirely is the master/slave pattern, widely used before the model came about. A process or a node is chosen as the master, then any work is distributed to slaves which produce a result and return it to the master. The MapReduce is based on this model in some ways, being that it has a master and slaves which process the tasks. However, in the MapReduce model, some slaves reduce, some map. In a regular master/slave model, all slaves process any task given to them and return it directly to the master.

This model is the basis of many modern models, though it is still used in the original form too. New models simply build upon it to add further advantages and remove limitations, such as the fact that it has a single point of failure by default.

2.7.1.2 Producer-consumer

Based on the master/slave pattern, this involves loops. Rather than simply having a master and several slaves, there are multiple producers and multiple consumers looping, passing data between each other. This is very handy when developing massively parallel applications, however it does introduce issues such as deadlocks.

Since this pattern uses queues, it is often possible to process work or keep results in a specific order. This is near impossible to do with the MapReduce model due to the fact that any worker could finish first. Additionally, the results in the MapReduce model are automatically sorted, not according to the order they were originally in.

2.7.1.3 Shared Memory

Shared memory models are a very reasonable alternative to the MapReduce model. In fact, there are even MapReduce implementations built to run in a parallel, shared memory environment [13].

Rather than distributing tasks across a cluster of machines, they are executed in parallel on the same machine, using the same memory. This works by making heavy use of threading, executing each instance of a task within a separate thread.

Shared memory models have been used long before MapReduce was introduced as they do have their benefits. One such benefit is the fact that all threads have access to the exact same memory, the same data. However, the performance when dealing with huge amounts of data is questionable considering a very high performance machine would be needed.
2.7.2 Uses of MapReduce

The MapReduce model is ideal for use in many different cases, just like this project as the implementations of it provide a general framework to build upon. Google, for example, uses MapReduce within their search service [13] to process terabytes worth of data per execution. Their most significant use of it is for large-scale document indexing, specifically for their web search service. Such indexing involves a map function which parses each document and outputs a large amount of word-document pairs. These pairs are then sorted and grouped by word, resulting in a large inverted index which may be used for searching [19].

There also exist high level languages implemented on top of the MapReduce model to allow less specialised programs to be developed quickly and easily, usually in a SQL-like syntax. One of these is “Pig”, a high level platform which runs as an extra layer on top of Hadoop, maintained by Apache. This platform and the associated language was developed as a solution to the inability to easily write programs for MapReduce which have multiple stages, rather than the usual two [22]. MapReduce only allows two primary stages, the mapping and reducing of input, so this language improves upon that and allows you to use a SQL-like language for creating multiple stages. For example, having multiple mapping functions or reducing functions rather than just the two, allowing the data to be processed in a more specific way.

This project is not hugely similar to that of “Pig” and Google’s search indexing, but the general idea of wanting to process large amounts of data efficiently remains. I could use a higher level language such as the previously mentioned one but doing so could easily overcomplicate things. The overcomplication would be caused by the fact that I only need a single map function and a single reduce function initially, making an extra layer useless as the basic MapReduce model already provides this functionality. Additionally, in Google’s case, terabytes of data would be processed per execution, whereas in this project, a far smaller amount of data will be used, probably in the gigabytes or megabytes range. This is actually beneficial to the project, though, as it will allow execution to be completed faster than if the data was larger, meaning it will be much easier to test and retry after altering any code.

2.8 Hadoop

As previously mentioned, there are a few issues which exist when developing applications to process data, specifically across multiple systems. Initially, development of such applications would also involve large amounts of complex code to handle these issues [13]. Hadoop attempts to solve this problem by providing a framework on which these applications can be built, but already have the underlying functionality. This means the developers are no longer required to determine how to distribute tasks, data and how to handle faults as it is all already implemented in Hadoop.

Hadoop was chosen for this project primarily because of these existing features. A large amount of data will be processed throughout the duration of this project, so distributing it correctly across
the virtual machines being used is very important. Ideally, we would like to have each chunk of data stored and processed on the same system, meaning all data is processed locally but to one specific node. This is exactly what Hadoop allows us to do, removing the need to worry about implementing such functionality.

2.8.1 History

In the past, prior to the MapReduce model being introduced, programs generally followed the master-slave design using some form of a divide-and-conquer algorithm. This was very similar to the idea of MapReduce in the way that it makes use of slaves, but it did not use the map/reduce model. Another approach people often used was the producer/consumer concept, involving a set of “producers” which output to a set of “consumers”. This could repeat for as many consumers/producers are required, but will ultimately result in some output.

Before Hadoop existed, there were very few implementations of the MapReduce concept, the most well known being Google’s [13]. The developers at Google had noticed that most implementations they had created to compute large amounts of data followed the same general idea. They all seemed to involve applying a map operation to each record of input data and a reduce operation to combine the data [13]. Given that most projects involved this same common functionality, it seems obvious to create a framework to handle it rather than re-implementing it each time. This is another reason why MapReduce is extremely useful for this project as it will involve heavy usage of map and reduce operations.

Hadoop was created from the idea which Google presented (MapReduce) as their own implementation was, of course, closed source, meaning an alternative was needed for the general public. The Hadoop framework provides a distributed file system with high reliability to control data management and a MapReduce implementation to analyse the data [26].

2.8.2 Design

Hadoop is designed in such a way that a few different nodes exist, each with their own task. The general design of Hadoop consists of a master/slave architecture [11] in which there are “DataNodes”, “TaskTrackers”, a single “NameNode” and a single “JobTracker”. The “NameNode” manages the file system’s namespace and the access to it [11] while the “JobTracker” manages which “TaskTrackers” to give tasks. So clearly, the “TaskTrackers” and “DataNodes” will handle the actual tasks and form part of the distributed file system.

Within this project, both the “JobTracker” and “NameNode” will be kept on the same host to reduce complexity within the system. Additionally, each slave will act as both a “DataNode” and a “TaskTracker”. Though generally, this is normal practice as we want the data to be stored on the same node which is going to process it. The reason they still remain separate within a system is because we may just want a distributed file system, without the processing features (MapReduce).
2.9 Related Work

There have been very similar projects in the past, both for research and for production use. A production example being the most obvious one, Google and their search engine functionality. Also, less common setups have been investigated such as running MapReduce over a huge set of idle computers [17] and for analysing scientific data [16].

A good example of related research would be the evaluation of MapReduce on multi-core and multi-processor systems [23]. This paper looks into how well the MapReduce model performs across systems which have multiple processors and/or cores, while also investigating the difference between MapReduce and alternative, lower-level models/concepts. One major difference between this research and my research is the fact that they use their own MapReduce implementation, “Phoenix”. Their implementation of MapReduce uses shared memory rather than multiple systems, meaning it specialises in parallel processing on a single machine rather than multiple machines [23]. Their evaluation of this implementation determined that the speed up was considerable larger as the amount of cores in use increased. Compared to sequential and alternative low-level implementations, it was much faster, primarily when dealing with applications which natively use a key-value approach.

Another similar piece of work involved developing an application for processing system logs produced by Hadoop and analysing them [24]. The basic idea was that logs generally contain useful information put in place by the programmer(s), not just numerical data. So it would be useful to have the ability to analyse this data and determine some outcome from it, such as an overview of program execution or data-flow.

2.10 Summary

This chapter has looked in depth at related work, technologies being used and in depth research into relevant topics. The next stage, following this background research and making use of it, is the design. The use of the map and reduce tasks, along with the use of any technologies must be researched.
Chapter 3

Design

3.1 Introduction

In recent years, the amount of data people deal with on a daily basis has increased massively. Businesses often accumulate terabytes of data each day, especially those primarily involved with web services, such as Facebook, Google and Microsoft. Essentially, the problem being tackled here occurs when someone decides they want to process this data. Processing such large amounts of data can be incredibly resource-intensive and will almost certainly take a huge amount of time when done in serial.

3.2 Methodology

The data used in this project was found either by Internet research or it was supplied by a third-party. It was then interpreted and analysed as required, though most or all of it was human readable anyway. Due to it being in a human readable format, there was little analysis needed, only an understanding of the data was needed.

As for experiments in this project, all were decided upon based on both related research and common sense. Developing an experiment to test performance of a system is often a rather simple procedure, especially in this case. Essentially, only a single application was needed to stress test the system(s) and tools needed for monitoring. The tools used were all chosen for being included in Linux by default.
3.3 Example Problem

Imagine a situation in which a company has a web service that receives hundreds of thousands of requests each day. Now, if each of these requests were to be written to a log file, the file would easily grow beyond a terabyte in size on a daily basis. Such a company may then wish to process these large log files to determine useful information. Maybe they would like to determine the most requested page for a specific country or the amount of users visiting a page from a specific referrer. Either way, it would take a large amount of processing power and time to determine such information.

In the past, the solution to this problem would be a bespoke application which processes the data in serial and has been tailored specifically to work with this particular data set. Although such a solution would work, it would introduce many issues which would likely be very bad for the company in question. The most obvious problem of these is that a huge amount of time would be required since there would be no parallelised computation. Additionally, it would execute on a single system, meaning a single point of failure. In terms of cost, it would cost a vast amount of money and would only work for this specific set of data.

3.4 Solution

Clearly, an ideal solution would be one which parallelises processing, is general purpose and implements some form of fault tolerance. Companies have been known in the past to have such a solution developed for them specifically, often lacking the ability to be used for multiple projects. This, of course, means a new solution must be developed on a per-project basis. It also means the code for handling distribution/parallelisation of the task, fault tolerance and all other common features must be written manually. This is where the MapReduce model comes in, allowing you to spend more time developing the actual application and less on the underlying system.

3.4.1 MapReduce Model

One of the best solutions available, though not the only one, is the MapReduce model. With this model and a valid implementation of it, only the application must be developed, not the underlying data handling, task distribution and such. In addition to this, the model is general purpose, meaning the system remains the same regardless of the tasks being processed. Only the applications used for the tasks need to be written, so costs will also be drastically reduced too.

Using this model, companies in similar situations to the previously mentioned example can easily process their data. Considering the fault tolerance, automatic distribution of tasks and management of data it implements, there should be no need for overcomplicated applications anymore.
3.5 Application of MapReduce

The application of MapReduce in this particular project was of great importance and was the primary reason for choosing the model as a solution. Both the map and reduce functionality of the model was used heavily within the project to create a realistic setup.

Although the model can be used without a reduce function, this project made use of both Map and Reduce to create a typical environment in which performance can be monitored. Of course the lack of a Reduce function would greatly reduce the amount of time used during execution and increase the performance overall. This would indeed be due to the fact that less code would need to be executed.

In a real world situation, a typical application involves grouping some data, sorting it and processing it. Combining these stages actually results in the MapReduce model, the map task groups and sorts while the reduce task processes.

3.5.1 Map

The map function of this model is primarily used to split the input into a set of keys while also sorting by those keys. If we want to process a large amount of data in a distributed environment, we of course want to be sure that the work is distributed correctly. Within this model, an entire set of entries with the same key are given to the same reduce task later on. As already discussed, each reduce task is executed by one node, it is never split between multiple nodes. Due to this behaviour, the choice of keys is a very important stage in the design of a MapReduce application. Imagine only two keys are ever made use of, this would create two very large, resource-intensive reduce tasks if each key has many values. This is exactly why the keys must be chosen carefully.

Within this project, map-only tasks were used purely for convenience or to test the general functionality of the system. Never were they used to measure any kind of performance as such tasks are often rather simple, too simple to be of any relevance. When processing data, map-only tasks will generally be used for preparing input data, possibly making it readable for another task. Another reason for doing so could be to sort the data beforehand, allowing for simpler map tasks in future processing.

Though the performance of the map task alone could be measured, it would be of little relevance without the performance of the overall task. For this reason, the performance was never measured in this project specific to one stage but rather it was measured overall, for the entire task. Often, some map tasks may also be executed in a small amount of time while the associated reduce task may take a lot longer. In cases like this, the performance of the map task alone would not show us much.

3.5.2 Reduce

The reduce function was used in almost every test throughout this project. Investigating the performance of the model overall, of course includes use of the reduce functionality. For any realistic
measurements of performance, a typical situation must be created, a common setup. Common applications make use of both the Map and Reduce tasks, which is why this project did too.

3.5.3 Performance

Measuring the performance of the MapReduce model was the entire purpose of this project, so determining the best way to go about doing so was very important. A few different things could be monitored to measure the performance, though only the most important of these were chosen.

The most straightforward way of measuring such performance was by making use of the execution time. Though it was a simple measurement, there were various different “execution times” such as the setup time, map time, reduce time and cleanup time. Each of these were taken into account but the total of all four was chosen as the deciding value for stating how well a task performed.

Disk usage was also decided to be used as a measurement of performance. By monitoring the disk usage on each node within the cluster, the distribution of data could be seen and the usage of each individual node’s disk. However important the CPU usage might be, it is equally important to take other factors into account to get an accurate overview of performance. Considering the strong relation between MapReduce and data, it makes perfect sense to measure such usage.

An additional measurement of performance would be the amount of CPU usage on each single node. The execution time and overall CPU time could be measured, but this would only show the performance in terms of speed. By taking the usage from each node, you can have a clear overview of where the tasks have been distributed to and which nodes are given most work.

Network usage could be another factor to take into account, allowing for the transfer of data in and out of each node to be monitored accurately. Such monitoring could give a clear picture of the data transfer which would yet again show the most active nodes. However, monitoring of the network was not a priority during this project as other measurements reached the same conclusions.

3.6 Summary

The design of this particular project was rather simple and straightforward due to it being mostly evaluation and testing based. As the aim was to investigate use of the model and the performance of it, less time was spent designing and developing applications than testing them. Also, due to unexpected issues or behaviour, the design and actual implementation could differ slightly.
Chapter 4

Implementation

4.1 Introduction

A selection of technologies must be made to develop a solution for this problem. This project will aim to investigate the performance of the MapReduce model, specifically on a cloud, making use of distribution rather than just parallelisation. So, given this aim, we need to choose the technologies to use wisely so we can maximise performance and monitor it correctly.

4.2 The Cloud

Publicly available commercial clouds have already been discussed and could well be used for research such as this. However, in such an environment, control would be restricted to that of the virtual machines and the layers above those. Since the aim is to investigate the performance, it would be beneficial to have control over underlying layers too. So, for example, it is possible to monitor performance of the physical machines. For this reason, the University cloud test bed will be used as it can be fully controlled and monitored.

4.2.1 Cloud Testbed

It was decided that the system to be used for this research would be the University cloud testbed. The availability of full control over this particular cloud allowed for effective monitoring of performance. Such control also allowed the environment to be setup in a specific way rather than having to use a system configured by a third party.
As you can see in the figure above, the cloud consists of several machines, each of which hosts a privileged virtual machine containing the hypervisor. In this case, the given cloud consists of 8 machines, one being the master which hosts the virtual infrastructure manager (OpenNebula) and the rest being child nodes. Although it is possible to launch virtual machines manually on each host, the logical way would be to allow the virtual infrastructure manager to automatically manage it.

The chosen cloud consisted of 8 similar systems, each of which had a minimum of 4GB of memory available for virtual machines to make use of. Though this amount of memory exists, it may not always be entirely free due to other users making use of it. For this reason, memory allocation will be kept at a minimum for each virtual machine in this project, while keeping enough to run tasks properly.

The head node which controls all other nodes has a larger amount of memory (16GB) as it is used when virtual machines are launched directly through Xen rather than OpenNebula. This node hosts both Xen and OpenNebula, the former being used for launching virtual machines locally and the latter being used for launching virtual machines on the cloud.

Without any additional configuration or special scripts, virtual machines launched through OpenNebula [6] are not persistent. This means that any changes made to the virtual machine while it is running will not be saved to the image, so if you launch the image again, it will not contain those changes and will revert to the initial state. If you launch a virtual machine locally through Xen [8], however, any changes will automatically be saved to the image when it is shut down. For this reason, all virtual machines would be setup through Xen initially, including the storage of any data being used. These modified images are then to be taken from Xen and launched through OpenNebula.

When creating an instance of a virtual machine through OpenNebula, a host will be chosen au-
tomatically for it to be deployed to. Generally, the chosen host is decided through a load-balancing type system, meaning hosts with the lowest load or least virtual machines are more likely to be chosen. This is very useful in common usage, but would greatly affect the performance in this particular project. The reason for this being the fact that most virtual machines will more than likely be placed on the same system(s), meaning there will be no real network usage between them when processing a task. Of course, network usage is a large part of the time taken for a task to be processed, so if it doesn’t exist, the results will be inaccurate and unrealistic. This is why the decision was made to deploy each machine manually to a host decided beforehand, rather than allowing it to be chosen automatically. Additionally, how close the data is to each host makes a huge difference as those closest to it will have a higher chance of being chosen.

4.2.2 Infrastructure

Many freely available toolkits exist for providing infrastructure as a service (IaaS). Given the choice of cloud, only the toolkit that particular cloud implements may be used, in this case the toolkit is OpenNebula. Though this may be true for this project in particular, there is no reason why any other toolkit cannot be used. Other popular solutions do exist and provide almost identical features, two of the most known being OpenStack [7] and Nimbus [5].

OpenNebula uses a templating system for defining virtual machines, a simple configuration file to put it simply. The template defines the name, memory capacity, network configuration and other such virtual machine attributes. This is an extremely useful feature for projects such as this one as it allows for multiple machines to be created from the same disk image. Any networking is configured automatically using the details provided in the templates, allowing for easy IP allocation.

As already mentioned, OpenNebula also allows virtual machines to be deployed manually to a specific host within the cluster. Such a feature is important when evaluating performance as it allows full control and means you know exactly which physical host has the machine.

4.2.3 Hypervisor

A selection of hypervisors exist, all of which are generally very good, though some do support more operating systems than others. The choice for this project came down to that of Xen or KVM [4] (Kernel-based virtual machine), both of which are extremely good and well supported hypervisors. Each has advantages and disadvantages, such as KVM’s use of native full-virtualisation whereas Xen uses paravirtualisation. Xen is a far more mature project than KVM and is known to have better performance, so the decision was made to use Xen [15]. The obvious reason for such a decision being that the performance is to be monitored for a typical setup, so using the same hypervisor most people use makes more sense.

Additionally, it is already known that full virtualisation has an increased amount of overhead compared to paravirtualisation. This is mostly to blame on the fact that everything is emulated in fully virtualised environments, meaning direct disk access and such is impossible, instead another layer
must be passed through first. Clearly high performance would be very beneficial, so, for these reasons, it would be the best idea to use paravirtualisation and have lower overhead. The only disadvantage of this choice is the fact that for a guest to run in Xen’s paravirtualised mode, it must use a modified kernel which is tailored to do so. However, this should be little or no problem at all, given that the correct kernel is available.

4.2.4 Limitations

Although the chosen cloud does have many benefits, it also comes with certain limitations which may well affect projects and research such as this.

The cloud consists of only 8 physical machines, rather small compared to the likes of Amazon and other public cloud providers. This does not matter much when measuring the performance with small amounts of data, but may be a problem when using large data. For example, if the data is at such a large size that it will utilise each of the 8 machines at once, it would be impossible to get realistic figures on performance. This being due to the fact that the maximum performance will always be reached and further progress cannot be made until additional nodes are made available.

Being that control of the cloud is shared, problems may arise when actions from other users conflict with your own, this is also a limitation of sorts. There is a risk that another user could, at any time, alter or control your virtual machines without permission. This is of course a limitation as true full control cannot be gained, meaning the performance cannot easily be measured without some other user affecting it at some point. A simple example of this is the idea that you may wish to monitor disk performance on a physical machine during a job, but then another user launches a virtual machine which uses the disk heavily.

4.3 Data

4.3.1 Introduction

As the aim for this project was to investigate the performance and behaviour with large amounts of data, a suitable source was needed. Of course, the data had to be of a suitable size, preferably a few gigabytes at least. Additionally, the data had to have a consistent enough format to be used as is, with as little further processing as possible.

4.3.2 Fault/Failure Data

Of the available choices, one was a large amount of fault logs from several clusters of machines [18]. In terms of size, this data was very reasonable, providing various logs of various sizes. Although none of the provided logs passed the 10 gigabyte mark, they were still a good enough size to use within the experiments used in this project.
Structurally, it was rather consistent and well defined. This particular data source was provided in a comma-separated format, allowing for easy parsing by any chosen language. Most languages, if not all languages have the functionality to parse a comma-separated file so this was ideal.

A sample of this data can be seen in Appendix D.

4.3.3 Weather Data

Many weather data sources exist online, available free of charge for research use. This can include rainfall data, climate data, average temperatures and so on.

Most of these data sets appeared to be provided in XML [9] format or a simple comma-separated format. Again, most languages can parse both XML and comma-separated data with ease, using native libraries. This was actually a far better structured source than the fault data mentioned previously but was harder to find reliable sources for.

The size of the data varied depending on the source, though most provided sets up to a gigabyte in size. This would have been adequate for smaller experiments but would contradict the aim of this project, to use large amounts of data.

4.3.4 Conclusion

For this project, it was decided that the failure data would be used rather than the weather data or any other alternatives. The reason for this, as already mentioned, was primarily due to the overall size. The failure data provided much larger data sets than the weather data. Size being an important factor in this project, meant that it would be logical to make this choice.

Additionally, the failure data had a much simpler structure to that of the weather data. Though more complex structures may be useful for parsing accurately, they do introduce overhead and more complex data processing.

4.4 Virtual Machine Setup

As mentioned previously when discussing the testbed, I decided to give each virtual machine a low amount of memory initially. The amount of memory would most probably have no effect on the performance if all machines were identical in specifications, meaning it can be increased at anytime without causing results to become inaccurate.

The first two virtual machines were each given their own dedicated disk image while any additional machines shared the same image. As said before, when the virtual machines are launched through OpenNebula, they are not persistent, meaning all changes are lost on shutdown. This is exactly why the master and first worker were given their own images, so any required data and applications could be stored on the images themselves. This way, when booted in OpenNebula, the data and applications will already be there rather than having to be copied across each time.
Any additional machines require a standard, unchanged installation with no additional software or data apart from Hadoop. Being that there is nothing specific to any particular machine in this situation, we can simply use the same disk image for every worker. This does, however, mean that any additional data to be processed must be transferred over each time the machines are created.

4.4.1 Operating System

Often a wide range of operating systems are supported for virtual machines which run on a cloud, including the common ones such as Windows and Linux. This completely depends on what the hypervisor being used supports, though most popular hypervisors have full support for the majority of popular operating systems.

Between the two major operating systems, Linux and Windows, the choice really has to be the former as Hadoop has limited support and very little documentation for the latter [1]. As for the distribution of Linux, it really makes little to no difference, only the installation may differ slightly but the overall system will be almost the same. For this project, Debian is to be the choice due to it being very well documented, completely free, largely supported by available software and having an extremely stable release. In addition to the stability and support Debian provides, it is also supported officially by Hadoop, with packages being released for every new version.

4.5 Hadoop Setup

![Hadoop Structure](image)

Figure 4.2: Hadoop Structure

The Hadoop installation used was initially from a Debian package installed using the package manager. Although this allowed for an extremely easy installation, problems were encountered when attempting to run the Hadoop daemons. The biggest problem was that the Debian package did not correctly setup any required system users or permissions. For this reason, I went on to remove the package and install manually from the source as provided by Apache. Particularly for this project, using the source package is a much better idea as you can easily customise the installation and control every aspect of it.
There is a way to have a separate secondary NameNode to act as a sort of backup for any file system information. Such a node would not store any data but rather the location of blocks, nodes and so on. Although this may be extremely useful in production setups, it is not really needed in this particular project. It only really affects the speed of the distributed file system and only by a very small amount of time, not the MapReduce implementation or the tasks running on it.

Each child node was chosen to act as both a DataNode and a TaskTracker. The primary and most obvious reason for this choice being the fact that we want data to be close to the task. If some nodes were to only act as a TaskTracker but not a DataNode, they would be forced to use data from another node rather than locally, meaning the data would not be close. This would indeed introduce many problems in relation to performance and overall execution time while also causing some nodes to be selected for work more than others.

4.6 Hadoop Distributed Filesystem

The decision was also made to use Hadoop’s own distributed file system for storage of the data being processed. HDFS goes hand in hand with Hadoop’s MapReduce implementation so it makes sense to use both together as they are even packaged together by default.

Technically, it is possible to run MapReduce jobs without the use of HDFS but this would not be recommended. The most obvious reason for not recommending such usage is the fact that data will not be distributed automatically or efficiently, rather it will need to be accessed remotely, manually by each node. Also, ideally we want a common setup rather than a customised or unusual one so the results we get in terms of performance are more relevant overall. Such a Hadoop setup would also mean the programmer would be required to manage data manually, such as controlling the distribution, replication and transfer.

4.7 Data Pre-processing

Before beginning any actual implementation work such as the Hadoop applications, the data needed to be understood and prepared. The fault log data being used was initially in a comma-separated format, natively readable by Python, the selected language for this project. For small, simple applications, the original data would be fine as it can easily be parsed and specific fields can be read from each entry.

For more complex applications or applications which specifically make use of the dates in these logs, the data must be processed beforehand. The dates in the logs were very inconsistent at times, so if they were needed, the original data would cause problems. The best solution to this problem would be to create a Map-only application which converts the data to a more natively readable format such as JSON (Javascript Object Notation) [3]. As for the dates, the recommended way to deal with such inconsistencies would be to parse and convert each date to a unix timestamp.

The data could be processed “on the fly”, just the same way it would be if processing it beforehand.
The same output would be created and the same functionality achieved. However, the reason for not doing this is, as expected, the performance. Processing each entry in the data every time the task is executed would certainly cause a huge decrease in performance, especially if multiple tasks are executed on a regular basis using the same data source. If the aim is to reach the highest level of performance possible, which it usually is, the best idea would be to pre-process data so it is done one time rather than each time the task executes.

To parse the dates within each entry, regular expressions were used along with Python’s ability to parse such dates from a given format. Regular expressions were used to determine which format the date was already in, then if the format was found, it would be parsed. However, if a format was not found, the entry would either be skipped completely or remain with an unreadable date.

![Figure 4.3: Example Data Manipulation via Pre-processing](image)

As shown in the above example, the pre-processing simply manipulates the data, it does not add or remove anything. Although nothing is really changed, the entries are actually ordered by their key as with any map task. The fact that the input for other tasks was then already sorted may well have caused some sort of micro optimisation. Although it may have helped other tasks, it is unlikely as the data is changed again and reordered regardless.

### 4.8 Measures of Performance

The aim of this project was to investigate the performance of MapReduce, so of course, methods of measuring performance were needed. Luckily, monitoring the performance of a single machine is identical to when dealing with a virtual machine. This meant that any methods used on the virtual machines could also be used to gather statistics from the underlying physical machine.

For most of the statistics, the Linux utility “vmstat” was used. The reason for this choice was primarily because all Linux distributions contain it, meaning the physical and virtual machines were guaranteed to support it. Of course, some other utility with more statistics or advanced features could be used such as Cacti [2]. However, using such a third-party utility would involve having to set it up on each machine manually. The CPU usage, disk usage and other common statistics could be retrieved at intervals, allowing for constant monitoring throughout experiments.

Gathering information on the performance of the actual MapReduce task was also a priority. Quite a lot of statistical information regarding resource usage could be gathered, however a higher level view on the task itself was needed. Higher level, meaning information such as how long the task took to
execute, how much CPU time was used across all nodes and how many nodes were utilised. All of these were actually provided by Hadoop “out of the box” by making use of the provided web interface.

4.9 MapReduce Applications

4.9.1 Introduction

The only accurate way to measure the performance of MapReduce is, of course, going to involve execution of various tasks. Several applications were developed for this exact purpose, though only two were used in the end. Of the applications which were created, a simple one and a complex one were used to allow for comparison of performance with different task complexities.

The efficiency of these applications was never really an important factor as their purpose was purely to stress the system. In a real world case, the applications would also be monitored and improved to maximise overall performance. To measure the performance of MapReduce alone, however, the applications and their purposes are not so important as long as they remain the same throughout all tests for a fair result. Although this is the case, they still did produce valid, useful output from the provided data and could well be used in production environments.

4.9.2 Simple Application - Total Uptime

The first step was to create a simple application which processes the chosen fault logs to produce some useful result. The simplest result to produce would be some sort of numerical data, such as totals, averages and so on. Given this fact, it was decided that the first application would simply total the amount of downtime per node.

As each fault entry already has an associated number specifying the total downtime, the application need only total these numbers on a per-node basis. In terms of complexity, this is at most one loop with \( N \) iterations, where \( N \) is the number of entries in the input data.

As you can see from the pseudocode, both the map and reduce tasks are incredibly simple. Essentially, they are the equivalent of a summation but distributed rather than executed in serial. It is expected that such a simple application would not fully utilise all available resources. However, this is what the more complex application was written for, while the simple one remains as a way to test the system in general. By testing a simple application on the cloud, the utilisation and performance can easily be compared to that of a more complex task which uses far more data.
4.9.2.1 Map Task

```python
import sys, simplejson as json
for line in sys.stdin:
    line = line.strip()
    fields = line.split(',')
    try:
        values = {
            'month': fields[16][0:2],
            'year': fields[16][6:8],
            'downtime': int(fields[18])
        }
        print '%s-%s\t%s' % (int(fields[0]), fields[5], json.dumps(values))
    except ValueError:
        continue
```

As you can see from the source code, the output of the map task would be a set of downtime totals on a per-node basis. Each outputted entry would represent a single fault, just like the input data. However, the difference here is the key. Each entry in the output has the key of the node it is associated with, meaning each node’s set of tasks would be processed by the same reduce task. Once again, as with the data pre-processing, the output was encoded in JSON (Javascript object notation) to reduce code complexity.

4.9.2.2 Reduce Task

```python
import sys
currentNode = None
currentDowntime = 0
for line in sys.stdin:
    line = line.strip()
    node, downtime = line.split('\t', 1)
    try:
        downtime = int(downtime)
    except ValueError:
        continue
    if currentNode == node:
        currentDowntime += downtime
    else:
        print '%s-%s\t%s\t%s' % (int(fields[0]), fields[5], json.dumps(values))
        continue
```

Figure 4.4: Map/Reduce Process for Total Downtime
The reduce task, again rather simple, sums each entry for each distinct node. It would take each set of downtime values for each node and calculate the total to be output.

### 4.9.3 Complex Application - Average Time Between Failures

After running tests using a simple application, the next stage would be to use a complex application. An application which ideally utilises most of the available resources and takes a noticeable amount of time to complete. Although a complex application is needed to test performance, it would not be a good idea to simply develop the most efficient one possible. Instead, the aim was to develop an application which was fairly efficient while also being complex and resource intensive.

The chosen application determined the average time between all failures, across all nodes in all clusters. This clearly means the expected result is a single integer representing the mean, although the application could well be modified to produce such a result per cluster.

The calculation in general is rather simple and could be performed easily iteratively in serial. However, the fact that it must be distributed means the problem becomes a little more complex. Due to the nodes being isolated from each other, some sort of overlapping would be needed to take into account the time between the last fault one task has and the first fault another task has.

The data was mapped in such a way that the output was a set of calculated keys and times when faults began. The keys were simply calculated by incrementing from zero until a certain amount of entries had the same key. This way the values should be evenly distributed as there was no key already available to group by.
As for the reduce task, it was made to calculate the average of the time between the faults it was given. It would then output the average, the number of faults in the set it was given, the first value and the last value. The first and last values were output as part of the overlapping, so the third and final stage would calculate the missing averages and calculate the final result.

4.10 Summary

Given the design and implementation which have now been explained in depth, the next stage is to test and evaluate the implementation. The primary aim of this project is to investigate the performance of such an implementation, which is exactly what the next chapter will do.
Chapter 5

Evaluation

5.1 Introduction

This chapter will primarily look at the results and performance of the actual MapReduce applications which were developed. Various experiments were created to determine such results and statistics, each of which will be discussed and explained.

5.2 Plan

Investigating the performance of the MapReduce model on a cloud is the primary aim of this project, making the evaluation the most important part of it. This plan will essentially explain how the model was evaluated and what applications or tests were used throughout.

Various different experiments were considered, each of which had a specific purpose to help determine the overall performance and functionality of MapReduce. The first and least complicated experiment was used for the sole purpose of testing the system functionality and behaviour. In this case, specifically the behaviour in terms of task distribution and utilisation of resources. To understand the performance of MapReduce, you must first understand how it works and how it behaves with different data set sizes. The second and third experiments were created to fulfil the aim of evaluating the performance with large data sets. An understanding of how MapReduce performs when dealing with large data sets was needed as the aim of this project was to investigate such matters.

In addition to the evaluation of the actual model, the implementation of it (Hadoop) will also be reviewed. The advantages and disadvantages of Hadoop will be looked at as it may well be a worse choice than other alternatives for future work.
5.3 Experiments

5.3.1 Introduction

To understand and measure the performance of the MapReduce model, several experiments were created. For each experiment, a job was executed across varying numbers of nodes. In most cases, this involved running the specified task on 2, 4, 6 and 8 workers, purposely one virtual machine per physical machine. The reason for requiring that multiple physical machines are used is mostly due to the fact that network communication needs to be taken into account for realistic performance. If all virtual machines were on the same physical host, the performance would definitely be affected and made inaccurate compared to real world setups.

For these experiments, fault log data from several clusters of systems was used, ranging over 10 years.

5.3.2 Common Setup

Each experiment used, at minimum, the same setup and the same number of nodes. The number of nodes used, unless otherwise specified, was 2, 4, 6 and 8 in that order. These numbers were chosen as the performance and difference between one and two nodes, for example, is not going to be as clear as between two and four nodes. So, in theory, having such equally spaced numbers of nodes will result in a clear performance change each time. As for the data, all experiments made use of the pre-processed data which was mentioned previously unless otherwise stated. It was also decided that the data would be stored on the distributed file-system (HDFS) to comply with common practice. This also allowed for tasks to access data as if it were stored locally rather than having to manually retrieve it from a remote location.

5.3.3 Experiment 1

5.3.3.1 Objectives

Using the processed fault data, the objective was to find the total amount of downtime for each cluster which had been referenced. This was the simplest of all experiments due to the fact that it was essentially a summation of some already available values. The data was already processed as discussed earlier, meaning it could be parsed with ease using native functionality provided by Python.

5.3.3.2 Data

As with every experiment, the processed data was used as input to the initial Map task. However, the only relevant information needed was the downtime each fault caused, the amount of time it took for the fault to be resolved, essentially. This simplified the application by a huge amount as the reducer need only deal with single integers as input values, rather than a more complex data structure such as an array.
5.3.3.3 Setup

The setup and preparation of this particular experiment was quite straightforward due to the simplicity of the actual application. Any data to be used from the processed output had already been stored on the distributed file-system, meaning it could be accessed without any prior stages of preparation. In terms of actual execution of the given application, Hadoop's streaming functionality was used to remove the need for Java-only code. Streaming the task rather than embedding it within Java allowed the application to be written in Python, a much simpler language. When using a Java-based task, Hadoop simply calls the methods for Map/Reduce with each entry while with a streaming task, it sends the data as input to the application. This also meant part of the preparation for the experiment involved writing the application to handle Hadoop's input correctly.

5.3.3.4 Results

As you can see from the results above, in terms of execution time there is little or no change while more nodes are added. The execution time does indeed drop by a substantial amount when 3 nodes are used rather than 2, however, it proceeds to increase again when a third node is made available. This is more than likely due to one of two things, or quite possible both. The first possibility is that the execution time became stable, meaning the change we saw was small and irrelevant. While the second possibility is that in one of the few executions, a task took longer than expected for some unknown, unexpected one-off reason. Only two executions were made on for each different number of nodes.
so it is quite possible that a randomly slow task caused the average to be inaccurate.

It seems the most obvious reason for the unexpected increase in execution time was actually due to the small data size and the small sample size. Given that the time changed by only a very little amount each time a node was added, it became apparent that the data size was likely the primary issue. Adding further nodes at this point would have had little or no impact at all. Such additional nodes would not increase or decrease the execution time by enough to be considered important. The reason for this behaviour is simple, the small size of the data being used, combined with the small size of the output means two workers would only ever be needed, if that. No matter how many additional nodes were added, the time would have remained roughly the same as only a very small amount of resources would need to be utilised, leaving most nodes idle.

An additional reason for the behaviour of the execution time was simply the way Hadoop works. Hadoop tries to keep the data and task close to each other, meaning if the data happens to be stored on the first two nodes only, tasks will be distributed to those nodes before anywhere else. For the additional nodes to be utilised, the first two would need to reach full capacity so the only obvious way to achieve this would be to use more data.

These results had no error at all, likely due to the small data and application size. As such, no error was taken into account.

![Figure 5.2: CPU Usage Per Node for Exp. 1](image)

The CPU usage as shown above clearly proves and agrees with the task distribution as discussed previously. The master node appears to have a spike in CPU usage, likely the initial setup and distribution of tasks, as expected. After tasks had been distributed, you can see the CPU usage of the master drops while node two increases drastically. This confirms the assumption that the task was
being distributed only to one node, leaving the rest almost completely idle.

5.3.3.5 Issues

This experiment went fairly well but, as always, there were some slight issues encountered throughout. Luckily, none of the issues were really of great importance as they did not alter the conclusions gathered.

One issue which has already been discussed to a certain extent was the inconsistent execution time of the final test. The problem was the sample size, only two executions were completed for each different number of nodes. As an average was used when reading the results, an unexpected slow task could have and actually did make the average inaccurate. This would be resolved by using a larger sample, but really such a change would not alter the conclusion so the issue was left as is.

An issue which did occur and was important in regards to all experiments was the inconsistency of system clocks. It was found that in some tests, the execution time seemed strangely slow as the map and reduce tasks had very inconsistent timings. For example, one task was found to be reporting a one hour map task while the rest took no more than a minute. Clock synchronisation was found to be the cause, all virtual machines needed synchronising beforehand in order to have accurate readings.

5.3.3.6 Conclusion

The conclusion of this experiment seems rather clean, small amounts of data are inadequate for testing the performance of the MapReduce model. As mentioned, this is due to the fact that data and tasks are kept as close as possible, as is network usage and such. Although this behaviour could well be specific to Hadoop, it is likely other implementations of MapReduce follow the same idea to maximise efficiency. Of course, it would be very beneficial if network usage, data transfer and overall load was reduced so this behaviour is actually very useful. The aim of this project is to investigate the performance, not to purposely cause bad performance, so this is a good conclusion.

5.3.4 Experiment 2

5.3.4.1 Objectives

The conclusions made from the first experiment showed that more data should be used. To fully understand the performance of MapReduce across multiple nodes, the use of more data is clearly a requirement. For this reason, the next logical step was to increase the data size and possibly introduce a more complex application to process it. In this second experiment, the task being processed was developed to determine the mean time between faults. Such an application requires far more processing power than the previous, simpler experiment so could be used to produce more accurate performance results.
5.3.4.2 Data

As previously mentioned, the data used in this particular experiment must be far larger than that of any previous experiments. So, this time, we used 1GB of input data rather than the previous amount which was less than 100MB in size. This would guarantee that the tasks would be distributed across more than just the first two or three nodes as experienced previously. The data was still from the exact same source as the previous experiment but was much larger and differed slightly in terms of format.

5.3.4.3 Application

The actual application being executed had the single purpose of finding the mean time between failures provided by the logs. Such a task may seem rather simple in serial but increases in complexity when parallelised. This increase in complexity was due to the isolation and separation of data, meaning no two nodes knew anything about each other’s data. No single piece of data knew anything about any other pieces, meaning only partial calculations could be made. The requirement for partial calculations also introduced the need for a single node to handle all the results to find the total mean. The easiest way to do such a “final calculation” would be a map-only task, a third stage added to the MapReduce process.

Obviously if you simply calculate the mean of each chunk, you miss out the time between a chunk’s last event and the next chunk’s previous event. To solve this slight problem, overlapping would be needed, such that each reduce task also outputs the first value of the data it has been given along with the results. This way, the final map-only task could use the provided value to calculate the missing time and correct the result.

It was decided for this experiment, a serial version of the application would be developed. This would then give an idea of the performance difference between a serial application and a MapReduce (distributed) application. The serial application basically used the same data but stored locally on the same machine it was being executed on.

5.3.4.4 Setup

Again, all data used during this experiment was stored on Hadoop’s own distributed file-system. This choice was made for obvious reasons such as data distribution across the nodes in use. Additionally, this application was yet again written in Python and passed through Hadoop using the included streaming functionality. Initially, two nodes were used, not including the master node and this number was incremented after two executions.

To monitor the disk performance during these tests, the Linux utility, “vmstat” was executed at set intervals throughout. This would give simple statistics in relation to disk usage, CPU usage and processes.

Each worker in the cluster for this experiment will use 512MB of memory overall, to be used by both the operating system and the Hadoop tasks. This may be too little of an amount as the operating
system alone will occupy most of it but this will be discussed in the results.

5.3.4.5 Results

![Experiment 2: Total Bytes I/O](image)

Figure 5.3: I/O for Exp. 2 on 2 Nodes
Figure 5.4: Map/Reduce Execution Time for Exp. 2

Experiment 2
Execution Time over Number of Nodes

Figure 5.5: Serial vs Distributed (4 Nodes) CPU Usage for Exp. 2

Experiment 2
Serial vs. Distributed CPU Time

37
5.3.4.6 Results Discussion

It is very clear from these results that the execution time decreases while the number of nodes increases. Initially, it was thought that the execution time would eventually increase due to an inappropriate amount of nodes. The reason for this thought was the idea that if the data size remains the same but the nodes increase, you will eventually be passing very small pieces between nodes, resulting in vast amounts of network usage. This network usage would then, in theory, cause execution to take longer rather than there being any speedup. However, this theory may have seemed like it would be almost certainly true but due to the way Hadoop works, it appears that is not the case.

Hadoop distributes the data and attempts to keep the tasks as close to it as possible. It does not simply split the data and tasks by the amount of nodes there are. Instead, Hadoop sends tasks to the nodes closest to the data first, until they reach capacity. At this point, it will then move on to other nodes, quite the opposite of using all nodes at once. Due to this, it is highly likely that some nodes will get very small amounts of work, if any at all, which explains why the time should never increase with the number of nodes for data of this size.

Looking at the execution time, we can clearly see that there was little to no change at all in regards to the cleanup and setup. Although this may not be the case in all situations, it is highly likely it will never take long enough to be relevant at all. The fact that the reduce task’s execution time drops at a higher rate than the map task is likely specific to this application. Some applications may have very complex map tasks, others may have complex reduce tasks, the difference will change per application.

Some map tasks appeared to fail during the duration of this experiment due to a lack of available memory. This problem was expected but did not cause so much of a problem that results were inaccurate or less efficient than usual. Of course, this problem can easily be avoided in future by allocating a larger amount of memory to each of the workers.

The disk usage did give some very interesting results too, as shown in the graph of the total bytes in and out. It appears after each spike, there was a steady, consistent amount of disk usage. The spike was likely the event when a node was given data to process, while the more stable usage was task sending data out. In this case, the data which went into each task was far larger than the data which came out, agreeing with the results here. It can also be seen that the master node (Node 0) had very little disk usage, this being due to the distribution of data to child nodes rather than the master.

A very important part of this experiment was the execution of a parallel version of the application to determine the difference in performance. Surprisingly, when executing the serial version of the application with the exact same data, it took only 5 minutes to complete. Compared with the MapReduce version, this was rather fast as the results show the MapReduce task took at least 7 minutes overall.

The parallel application seemed to use far less CPU time than the distributed. For the distributed application, the total CPU time is a total of all time taken by all nodes, though in reality, this was in parallel. Due to this reason, an average was used to give a better representation in the results. It becomes clear that the distributed application used slightly less CPU at any given time but took longer overall to finish.
The results had little or no error in terms of performance monitoring, which is why no such error was taken into account within the results.

5.3.4.7 Issues

There were very little issues, if any, involved in this particular experiment. Luckily, no major issues existed, meaning the results were reliable and accurate.

There were times when a small amount of tasks failed, usually the reduce tasks. The reason for this seemed to differ each time, making it likely to be a random unexpected problem. In most of the cases when such an issue occurred, the task was executed again and the new results were used instead. Executing the task again would be the obvious thing to do as failed tasks could make results inconsistent. If one execution had several failed tasks while another had none, the former would have spent time resolving the problem, meaning inaccurate results.

5.3.4.8 Conclusion

This experiment shows that the use of data distribution is very important when attempting to maximise performance. Throughout the entire experiment, the master had almost no disk usage at all while each of the nodes spiked regularly. If the data was stored on a single node, such as the master, there would have been a lot more disk usage overall on each node. This usage would have been caused by each node having to fetch data from the master, meaning the master would always have some disk operations going on. With a distributed file-system however, each node need only access local data or blocks from other nodes, never from the master.

From the results of the serial execution, it can also be concluded that not all applications are suitable for the MapReduce model. In this case, the reason for serial version being faster was likely due to the fact that the complexity was greatly increased within the distributed version. As it was calculating an average, the algorithm was much simpler in serial as all data was known by the application. This shows that some applications may become too complex when distributed, meaning they are better off when executed in serial.

5.3.5 Experiment 3

5.3.5.1 Objectives

The primary objective of this third experiment was to find if the performance changed and, if so, by how much when increasing the data size. Although the previous experiment did give clearer results than the first, a larger amount of data would use more processing power and create more stress. Increasing the data size to an even larger amount than previously was expected to cause the task to use all nodes rather than just some.
5.3.5.2 Data

Previously, 1GB of data was used as input and it was processed as expected in less than 15 minutes. In this experiment, the data was increased to 10GB rather than 1GB, much larger than before. This increase in data size was expected to cause the task to take much longer to process in comparison. Although the data was larger, it was from the same source as previous experiments and had the same format.

5.3.5.3 Application

The same application as the previous experiment was used in this experiment too, although it was slightly modified to improve distribution of the tasks.

5.3.5.4 Setup

Due to the increased size of the data, this experiment required some additional changes during preparation and setup. The virtual machines all used 10GB disks, meaning the actual free storage amounted to less than 9GB as the operating system occupied the rest. Being that the data was 10GB alone, this meant that at least the master node needed a larger disk so it could store the data initially to be distributed across the cluster. Once the data was distributed however, it could be removed from the master node.

Additionally, the amount of memory on each of the workers was found to be inadequate at 512MB, as mentioned in the previous experiment. This time, each worker was allocated 1GB of memory each, more than enough for processing the data involved in this particular experiment. The master still had the same 512MB of memory it had previously, this being primarily due to the fact that it did not process any of the tasks but rather managed the workers which did.
5.3.5.5 Results

Figure 5.6: I/O for Exp. 3 on 4 Nodes

Figure 5.7: CPU Usage for Exp. 3 on 4 Nodes
5.3.5.6 Results discussion

As shown in the results, all nodes except the master had roughly the same amount of CPU usage during execution of the task. This agrees with the expectations of all nodes being used due to the increase in data size. Clearly every node was active, processing something, meaning instead of the tasks being allocated to only the first two, they were distributed between the entire cluster. As per usual, the master node had a very small amount of CPU usage throughout the duration of the task as it only needed to manage tasks, not execute any.

However, none of the nodes were being fully utilised at any one point in time. For such behaviour, even more data would be needed or a far more complex application. Increasing the data would be the best option as the MapReduce model exists to simplify applications, we do not want to forcefully and purposely make an application overcomplicated. It is expected that when the nodes become fully utilised, the performance will stabilise as it would not be able to increase any further.

The disk usage does seem to have been very erratic rather than following some sort of pattern like in experiment 2. This may well be due to the fact that all nodes were being used but not continuously. The results of disk performance show that the first two child nodes had a large amount of activity while the others only had some occasionally. This was likely because the other nodes received far less work, most probably due to them being further from the data.

Again, there were was no error in the performance results gathered, meaning the presented results are accurate.

5.3.5.7 Issues

A huge issue was experienced when attempting to run tests using this experiment’s application. The job was found to be reaching roughly 50% completion then being killed due to too many failed reduce tasks. The first stage of resolving this issue was to trace the error back to the relevant log files on child nodes. In this case, the error was actually very generic and was not self explanatory at all. Often such issues are either due to memory issues or disk issues. Initially, it was thought the problem would be a lack of memory, so all virtual machines were allocated a further 512MB of memory and the experiment re-run. This did not actually fix the issue, so the next step was to increase the disk size of each node from 10GB to 15GB. The increase in disk size did indeed fix the problem, meaning on at least one of the nodes, the disk was reaching maximum capacity during execution.

5.3.5.8 Conclusion

This experiment concludes that, even if all nodes have enough storage to contain the entire input, they may still reach maximum capacity. The reason for this may vary between experiments, though in this case it was likely due to a node already storing a large chunk of the data on HDFS. In the case of that specific node, it could have filled the remaining free space when saving output. Since, of course, the input and output total to a larger size than that of just the output, meaning nodes need more than the
size of the input to be safe.

It can also be concluded that, as expected, more data is needed to be able to utilise all available nodes. Even then, though, the nodes are not fully utilised but rather only partially. Considering the data size in this experiment was 10GB and only two of the four nodes were used heavily, an increase to at least 30GB would likely be needed for full utilisation.

5.4 Hadoop

5.4.1 Introduction

While the evaluation of actual performance is key to this project, it is equally important to take into consideration the implementation of MapReduce being used. In this case, Hadoop was the choice for many reasons which have already been discussed, such as it being open source and widely supported. Everything has a disadvantage somewhere, either one that can be removed in future or one which exists to provide an advantage.

5.4.2 Issues

One issue lies within Hadoop’s distributed file-system, HDFS. HDFS could not be mounted like a regular file-system using native support provided by the operating system. Though this wouldn’t be an issue if only dealing with MapReduce tasks, it may introduce problems when using it separately. For example, a service may want to store data across multiple nodes, in which case the idea of HDFS would be perfect. However, as it cannot be mounted normally, the applications would need specialised implementations to interact with HDFS.

A rather large problem with Hadoop was the location and distribution of logs. When an error occurs during execution of a task, the error was logged on the node which was running it at the time. This means, unfortunately, the programmer must first find the node containing the log, then read it on that node. Having a central location for all log files would be a much better idea, although it would require more network usage and such.

Although these two issues do exist, Hadoop is still being continuously developed, meaning they may well be resolved in future.

5.4.3 Limitations

When designing applications to be executed by Hadoop, it was found that there was often a need for a second reduce task or a second map task. Hadoop does not provide the functionality required to implement multiple map or reduce tasks, so this could be seen as a limitation. However, this limitation is likely known and left as is on purpose. The reason for leaving it being the fact that if you need multiple map/reduce stages, you are likely not following the model correctly. In any case, this
need can be fulfilled by making use of two separate MapReduce tasks such that the second takes the first’s output as input.

A huge issue and limitation with Hadoop is the fact that it has a single point of failure, the NameNode. If this particular node fails in any way, the entire MapReduce cluster becomes unusable. It is possible to create a secondary NameNode, a node which stores copies of the master’s data. However, if the master fails, the secondary is only there as backup, not to take over the handling of jobs. This goes against one major reason for using a cloud, fault tolerance. When dealing with huge amounts of data, the last thing anyone wants is for the job to fail half way through because the master had an error. If this were to be fixed in future releases of Hadoop, it would make the implementation far more stable and reliable.

5.5 Minimum Requirements

In terms of meeting the minimum requirements, this project did quite well. Of the requirements chosen prior to starting the project, all of them were met to a good standard.

The chosen MapReduce implementation, Hadoop, was setup exactly as required and functioned perfectly well. It was used for all experiments in this project without many issues at all. The issues which were encountered were actually related to the applications and virtual machines rather than the Hadoop installation.

Several applications were developed to work with the MapReduce model and be executed by Hadoop. Of the chosen applications which experiments were created for, all of them worked well and produced an actual result. In the case of the second experiment, an actual figure was produced which could be used to determine which systems to use for hosting new virtual machines.

One of the developed applications was also compared with an equivalent serial version. This provided a realistic representation of the difference between serial and distributed performance. The comparison of the two showed that some applications perform better in serial while others perform better when distributed.

Overall, all minimum requirements were exceeded and developed upon throughout the experiments and project.

5.6 Methodology

The methodology of this project proved to be rather effective in meeting the minimum requirements. Research done before starting the project came in extremely useful for understanding reasons for behaviour within Hadoop and the MapReduce model. It also proved useful knowing which technologies are used as standard in the industry, allowing for the environment to replicate that of a realistic, production equivalent.

The original plan was to develop and implement at least one application on the University Cloud.
This and other applications were actually developed, each of which had a real use and could work in production environments. By developing such applications, most of the minimum requirements were met as already discussed. They allowed for good performance figures to be produced, similar to those of a real world situation.

5.7 Existing Work

Some similar work does exist which produced similar results to those found in this experiment. Many pieces of work do investigate similar matters but a large percentage of them are specific to some specialised setup/system.

One such similar piece of work involved implementing MapReduce in a volunteer computing environment, meaning making use of idle machines around the world [17]. The results produced by their implementation were consistent with the ones presented in this project. This specific work found that when data was only distributed across a few systems, the overall execution time increased. This is similar to experiment 2 presented here, the data was only distributed across some nodes so the execution time did not drop as much as it could have. Although in this project the execution time did drop, it was not by a substantial amount and would likely go further if data was distributed across all nodes.

Existing work has also evaluated the MapReduce model in depth using far larger data sets than those used in this project [16]. The findings produced showed that the execution time of a MapReduce task eventually stabilises rather than decreasing constantly. This is exactly the same finding presented by the experiments in this project. As already discussed, the reason for this is that additional nodes simply are not utilised unless the data is so big that previous nodes are at full load. The same work also found that the speedup of applications eventually stops increasing and, instead, remains at roughly the same point.

5.8 Further Work

This project did develop a good understanding of the MapReduce model and the performance of it. Many interesting results were gathered and many findings presented. However, due to the time limitations, several extensions could be considered to draw further conclusions.

One such extension would be to investigate the performance with much larger amounts of data. In this project, the maximum data size reached was 10GB, actually rather small in comparison to what companies typically encounter these days. To get a more accurate representation of performance on a more common scale, sizes in the terabytes should be considered. Such a large amount of data would create a much better stress test for the model and the implementation of it.

Another important extension to be considered is to use a much larger number of nodes. The experiments in this project used a maximum of 8 nodes, each of which was on a separate physical...
machine. There was a limitation due to the cloud consisting of only 8 physical machines. Although many more virtual machines could be created, they would be deployed on the same physical machines, making performance better and less accurate. A good recommendation for such an extension would be to use a minimum of 8 nodes, up to possibly more than 20 nodes. This would go hand in hand with the previous suggestion of increasing the data size. As shown in the experiments presented in this project, the task distributed does not grow with the cluster size unless the data grows too. So an increase in both data size and cluster size would allow for full utilisation of the nodes.

It could also be beneficial to investigate the use of other MapReduce implementations. A very interesting choice could be a GPU-based implementation to see if the performance is better or worse. Such an implementation would likely use CUDA [12] to execute code natively on the GPU. Of course, GPUs are highly parallel which would suggest that they would perform well for a model such as MapReduce. However, this is a conclusion to be drawn by such an extension as it may well turn out to perform worse. Another implementation to investigate could be one which runs purely in parallel rather than in a distributed environment [23].

One final suggestion for an extension to this work would be to test the performance on a public cloud. In this project there were many limitations due to the use of a very small private cloud. It would be an extremely good idea to investigate the performance of MapReduce on a public cloud such as Amazon’s EC2. In the case of Amazon’s compute cloud, EC2, Xen is still used so the performance results should be comparable to the ones presented here. The EC2 cloud makes a few different instances of virtual machines available, each of which has more memory than the ones used in this project [14]. It would be interesting to research the difference in performance between these instances as they have more resources than were available in the experiments here.
Chapter 6

Project Conclusion

6.1 Discussion

This project had the aim of developing an understanding of the MapReduce model, specifically on a cloud. Several applications were developed for this use and a couple were used as the basis of experiments. Each application was executed on varying numbers of nodes to compare the distribution of tasks and overall performance.

It was found that the MapReduce model, as expected, did reduce the execution time of most applications. However, through further experimentation, it was found that certain applications performed worse when using the MapReduce model and better in serial. It is well known that not all algorithms perform better in parallel due to a vast increase in complexity. Algorithms which involve the use of updated values are the best example of this issue. For example, imagine an algorithm which iterates over a set of values in serial. Per iteration, this same algorithm produces some result which takes into account all previous results. In this case, parallelising such an algorithm would involve a huge amount of overlapping or communication between processes/nodes to know the results they are producing. This is exactly why such algorithms would probably perform better in serial.

Another finding was that due to the way Hadoop in particular works, a large cluster would be excessive and unnecessary unless dealing with large data. This is actually a benefit of the Hadoop implementation rather than a problem. It is a benefit as small tasks require less resources and never distribute too much, rather they utilise the minimum number of systems first. This does, however, enforce the conclusion that MapReduce and the Hadoop implementation of it are likely not needed with small projects. Multiple systems and a more complex structure often results in higher costs, in which case it would not be worth it for small applications. However, in terms of performance, Hadoop
does perform very well, usually better than a serial application.

The overall conclusion of this entire project is essentially that the MapReduce model does work very well in most situations. Although it performs at an equal level to that of a serial application in small cases, it does not often perform worse. In addition to the performance, the functionality and implementation of the model provides a solid base to work on. When comparing the complexity of regular applications with MapReduce applications, it becomes clear that the ones built for MapReduce are far simpler. This simplicity is basically due to the fact that all of the underlying data handling and task handling has already been implemented. So, the model actually does do exactly what was expected, it reduces the amount of code needed, increases performance and makes applications scalable.
Bibliography


Appendix A

Personal Reflection

Overall, the project went very well and the aim was definitely achieved as I hoped. It has been one of the largest pieces of work I have done before and has involved far more research than anything else. Although it did prove difficult at times, it was worth it to improve my skills and knowledge on the topic at hand. Being that the topic was very interesting to me beforehand anyway, the project proved to be quite fun. I already had a good understanding of clouds and other such technologies, allowing for me to build on that throughout this project.

One thing the project did improve was my time management skills and how organised I was. I think it is extremely important in such a project to keep your own schedule and enforce it, stick to it at all costs. Considering the fact that the project had no predefined schedule or mandatory time in university, it would be very beneficial to create one for yourself. Regardless of how well you stick to schedules made for you, having to make your own is a rather big responsibility and the downfall of many people. It is very easy to lose track and fall behind without such planning.

Communication is another major requirement in projects such as this, I believe. In my opinion, it is very important to have regular communication with your supervisor to be sure you are going in the right direction, staying on track. There were many times when I simply needed to clarify a few things so I could be sure I was not going off topic. Without such communication, I think I would have definitely found myself stuck at several points in the project, confused about which direction to go in. Especially if you plan on continuing work at university and possibly having the same supervisor, it is very useful to get to know them well.

I think it was also very important to take into consideration other projects when working on a shared system. It may well be a good idea to use a cloud which not controlled by multiple people. I found that almost every problem which was found with the cloud was actually due to another student.
A good idea would be to take such things into consideration beforehand to help decide if to use the University cloud or not.

Additionally, I think it would be a very good idea to learn about the technologies and such beforehand. I personally already knew a lot about Hadoop, MapReduce, OpenNebula and any other technologies used in this project. This came in extremely useful as I already understood the systems and could, instead, spend time on developing the applications and evaluating them. Basically, it is a very good idea to research anything being used in the project before even beginning it.
Appendix B

Materials Used

Throughout this project I made use of several applications and technologies to simplify the process. Most of the technologies I used were actually required since developing a custom equivalent would be outside of the scope of this project.

I made use of the Xen hypervisor to launch virtual machines, each of which used the Debian distribution of Linux. In addition to Xen, OpenNebula was used to manage deployment of virtual machines on the cloud and allocation of resources.

In terms of programming, no third party implementations were used, all code was written for this project only. The systems used to execute these applications were third-party, though, such as Hadoop and Python. No Python libraries were used except the native, included ones.
Appendix C

Ethical Issues

Before beginning this project, an investigation was made into potential ethical issues associated with it. The result of this was that there were apparently no issues involved. This probably being due to the fact that all code was custom and will never be distributed but will stay private.

In terms of data, it was all free to use and contained no personal information and would be near impossible to use unethically. The data was completely generic, meaning the systems associated with the logs were not mentioned anywhere in detail.
## Appendix D

### Sample of Data

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
<tr>
<td>cluster</td>
<td>49</td>
<td>6152</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>5-Apr</td>
<td>5-Jun</td>
<td>current</td>
<td>part</td>
</tr>
</tbody>
</table>

Table D.1: Table of example data [18]