University of the West of England
BSc(Hons) Computing

Designing & Creating a Self-Managing, Distributed File System

Author:
David Bond

Supervisor:
Rong Yang

April 12, 2016

Abstract

In modern distributed file systems a significant amount of administration is required to ensure that the cluster is working properly at all times. Many systems do not provide a basic "plug and play" functionality where nodes can simply be added or removed from the system. This project aims to report the creation and development of such a file system.

Instead of user configuration the system aims to utilise evolutionary algorithms to make decisions on aspects of the file system such as file location and replication. These algorithms will be compared for efficiency and suitability to determine a best fit.

Once the system is developed it will be benchmarked to compare its operational ability with those of other distributed file system platforms (such as HDFS and GFS) to determine whether less user configuration is appropriate for these kinds of systems.

Acknowledgements

Dr. Antin Leszczyszyn, 7 Layer Solutions LTD.

I would like to express my thanks towards Antin Leszczyszyn, Big Data Scientist at 7 Layer Solutions LTD. for providing me with valuable aid in the production of my cluster and giving me assistance expressing my algorithms in mathematical notation.

Dr. Rong Yang, University of the West of England.

I also would like to express gratitude towards my degree head and supervisor Rong Yang for her support and continuous encouragement. She provided me with all the necessary resources and facilities required to complete my research. Her guidance has been enormously helpful and valuable.

Dr. Zaheer Khan, University of the West of England.

Dr. Khan acted as the second supervisor to my project and I would like to thank him for his time spent evaluating the report and providing me with valuable feedback.

Contents

Code Listings

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Project Aims

The aim of our project is to create a distributed file system where cluster administration is fully automated by the master and data nodes. New nodes (individual computers within the system) must be able to join the cluster using only the IP address of the master node. All basic file operation functionality must be available, such as adding, deleting and renaming files or directories. On top of this, the system must determine which nodes to send which files based on a weighting it calculates based on environmental factors of each node.

The approach we intend to use is creating a requirements document which fully describes the system intended and use it to develop the project in C#.

1.2 Beneficiaries

The main beneficiaries of the system will be those who use or want to use a distributed file system which requires low maintenance and subject knowledge, whilst still providing adequate data redundancy, file access speed and security. Most distributed file systems available now (such as Hadoop[2] or Cloudera[21]) require a lot of administration, management and potentially certification in their use. This project intends to create a system without these difficulties.

Chapter 2

Research

2.1 Existing Distributed File Systems

A distributed (or clustered) file system shares its files across multiple servers simultaneously. They also use file replication among nodes to increase data redundancy; a method to improve the reliability, reduce acquisition complexity and to promote the integrity of a cluster.[1]

Below are currently existing distributed file systems which are regularly used by software companies to develop solutions. We will explain them in detail and their shortcomings will be described and compared to the improvements of our system proposed in this project.

2.1.1 Hadoop Distributed File System

HDFS is a distributed file system contained as a package in the big data framework Hadoop. It is designed to run in a large variety of hardware environments. It shares its functionality with many modern day file systems, except its unique design to run on low-cost hardware and its ability to tolerate a high amount of faults within the system whilst still remaining functional.[2]

Although HDFS has become widely used it does suffer from many issues. Some of these include:

- HDFS is completely written in Java.

  Unfortunately, Java is a very vulnerable programming language which has, in the past, been heavily abused and exploited by cyber criminals. An example of this was in 2013 when a hacker used a 'zero-day' vulnerability which allowed remote users to run arbitrary code on a machine. At the time, security experts recommended permanently disabling Java.[5]

  The proposed system will use C#, which will avoid the security concerns that exist in the Java programming language.

- HDFS's architecture raises security concerns.

Because HDFS has to run within Hadoop, issues arise from the sheer size of the ecosystem. It is a complex application to manage and can sometimes require several certifications for a person to be allowed to manage the system professionally. Hadoop's security model is also disabled by default because of its complexity and it lacks encryption at storage and network levels.[2]

The proposed system, being custom built, will not require the large amount of knowledge needed for each of its individual components reducing the overall complexity and can also be easily modified. The proposed system can also be simply modified to encrypt network packets and any data stored.

- Hadoop is not well-suited to small scale operations.

  The design of Hadoop is very high capacity. This means it is designed to create solutions requiring frequent processing of extremely large files. This means that it is not appropriate for companies to use for analysing smaller data-sets.[2]

  The system proposed intends to be intrinsically scalable and to handle files of any size. Having an automated node addition system means that the hardware used can be as small or large as desired, allowing expansion if larger data is to be processed.

- HDFS is completely open-source.

  The Hadoop File System is written entirely open-source, meaning that functionality is created by many developers who contribute regularly. Although this means that many improvements can be made, there is no formal quality assurance for each developer, potentially leading to unstable code. Hadoop itself has had many instability issues, meaning problems may arise for HDFS users.[5]

  The proposed system will not be made open-source and will be developed using a strict style guide to ensure code coherency. It is also intended

that appropriate testing will be undertaken frequently during development to ensure bug-fixes are routine.

Ultimately, HDFS suffers from many issues which should not occur in the proposed project. However, HDFS has been in development for many years now and there may be unexpected issues which occur in the development of this project which already have been dealt with in a previous HDFS release.

2.1.2 Google File System

The Google File System (or GFS) is another example of a distributed file system. It is proprietary software developed by Google for inter-company use and development. It functions similarly to HDFS (discussed above) which focuses on utilising commodity/low-cost hardware.[6]

Google's version of a distributed file system shows a very well thought-out architecture with regards to the files it stores. Some bonuses of using GFS include:

- GFS is completely scalable with low-cost hardware.

  Like HDFS, GFS is designed to function with almost any hardware, allowing very simple scaling at a low cost. This is a double-edged sword however as lower quality hardware can lead to more faults for the system to tolerate, older hardware running new software will ultimately lead to faults if the system is not designed to support older hardware.

  The proposed project should run as just a basic C# application. Optionally, code could be compiled on each machine that intends to run it, ensuring that the application runs to the specific hardware, nowadays many linux-based applications require building on the system it is intended for them to run on.

- GFS provides fault-tolerance on a similar level to Hadoop.

  One of Google's greatest challenges was to create a DFS in which component failures are handled internally by the system[6]. In a large-

scale cluster, it is obvious that component failures will happen, and countermeasures must be appropriately put in place. Google created fault-tolerances for lapses in data integrity, auto-detecting corruption of data and re-replicating data across functional nodes.

Because of time constraints, the proposed system will not be able to provide fault detection on the same scale as the Google File System. However, it is intended that when a detection of a disconnected node occurs, file replication values are modified and data is moved accordingly around the cluster.

- GFS performs incredibly well on high-scale operations.

  Judging by benchmarking performed on GFS[6], the system was able to consume data varying around 101MB/s. It was also able to restore itself at an approximate speed of 440MB/s. This shows that clearly the GFS is an incredibly powerful system which fully utilises its hardware.

  Our project intends to follow-suit alongside GFS. An algorithm where nodes for data reading or writing are chosen by a weighting aims to reduce bottlenecks and make clustered actions faster. However, as the hardware available for the project is nowhere near the scale of Googles development cluster, we may expect slightly lower values in benchmarking than seen with GFS.

2.1.3 IBM General Parallel File System

In 1998, IBM introduced GPFS as a high-performance distributed file system. It is currently used worldwide by many different companies and is widely used among many implementations of supercomputers[7]. GPFS also contains many features which increase usability and maintainability. Examples of these include:

- GPFS Native Raid

  Also known as 'de-clustered RAID' (Redundant Array of Independent Disks), any RAID operations are performed at a different level to what

you would expect in a system such as this. Instead of performing operations at the disk level, it is instead performed at the block level. In the event of data loss, rebuilding the lost information using redundant data can take less time to perform using blocks as opposed to entire disks. This is especially prevalent as storage capabilities continue to rise, instead of having to restore, for example, a 1TB disk for just one file, GPFS will simply restore the missing blocks, reducing operation size[8].

It is intended that operations performed on files in the proposed system will be at the file level. Due to time limits it is not feasible to implement a block system which would separate and store file information. This should only have an effect on networking speeds and file manipulation speeds. However, as we are developing a weighting system to determine *where* files are stored, it is possible to use network latency as a deciding factor, potentially counter-acting the reduction of speeds due to file-level operations.

- Disk Hospital

  IBM developed this software in order to constantly monitor the health of disks within the distributed file system and to perform benchmarks on individual pieces of hardware. Due to this, hardware failures and failing disks can be found early enough to prevent permanent damage to data.[9]

However, despite the usefulness of features such as these, there are some issues with using GPFS. For example, the disk hospital plays a very important role, but does not take into account that within storage arrays, many perceived 'failures' can simply be prevented by re-flashing firmware or other simple solutions that do not require the cluster to enter a recovery state which can slow down operations on other nodes within the cluster.

Another issue with GPFS is that a personal network file system (NFS) gateway is required (which is no easy task on a large scale system), increasing the cost

of development and implementation. It also does not contain a NAS gateway, decreasing simplicity of management[9]. The lack of NFS and NAS (Network Attached Storage) means that running multiple clients simultaneously is relatively impossible. In comparison to Hadoop or GFS (described above), running a large amount of clients seems to be an important feature that other file system developers prioritise highly.

2.1.4 Summary

In this section we have discussed existing distributed file systems and evaluated their advantages and disadvantages. From this we can obtain features which are ideal for the proposed system. These are based on what we want to take away from existing systems and what should be added to them. Using this information we are able to produce requirements for a system that will run either as well or better than the previous systems discussed.

2.2 User Intervention in Existing Systems

In this section we will discuss the amount of user intervention required in order to successfully implement a clustered file system. In current times many of these systems require expert help or excessive certification in order to fully implement. However, some systems such as Ambari[11] (for use with HDFS) allow for the automated setup of the system and provide a much friendlier interface in order to manage the cluster. For example, by default HDFS requires the use of a command-line interface in order to manage and administrate, Ambari provides the user with a web-based interface which can be accessed via the master node's IP address.

In order to fully contrast and compare the amount of user intervention required in setting up and administrating a DFS using the aforementioned software we will be looking into the administration documents provided to compare how much needs to be done in order to set up and manage a cluster in each

framework.

2.2.1 Managing HDFS without Ambari

To install HDFS using online packages the premise is fairly simple. It relies on the user downloading and configuring the Hadoop packages using .XML and .SH files. The only prerequisite for installation is having a version of Java which supports HDFS.[11]

In order to build a cluster, one must install Hadoop on all hardware intended to be used by the file system. This can take a very long time to do if the intended cluster has a large amount of nodes. This issue could be worked around by using disk images, however some nodes will need to perform different tasks that require new configuration, making imaging useless in this case.

Having installed Hadoop on all required hardware the user then faces greater difficulties. In fact, Todd Papaioannou (CEO and founder of the company Continuuity now Cask, who set up Yahoo's Hadoop architecture) states himself that Hadoop is 'Damn hard to use'. In the architecture he designed it was intended that the platform had to be stable enough to allow 5,000 Yahoo developers to build applications on a daily basis. The main issue he describes as being that Hadoop is 'low-level infrastructure software' and that most people 'are not used to using low-level infrastructure software'.[12]

The main difficulty comes from the plethora of configuration options available to suit HDFS to a specific customer need. In fact, MapReduce (a package used by Hadoop and HDFS to perform most operations) has 219 parameters which can be tuned before even starting the software package itself[13]. This means that a huge amount of domain knowledge is required in order to fully 'fine-tune' a Hadoop system. Leading to development time scales which can exponentially grow as client need changes.

Once all of the parameters are configured the administrator can then modify the file-system at their own will using the command-line interface. Files can be added, removed and modified relatively simply with its performance being based on the individual node's hardware.

From these points, one can see that the set up and administration of the Hadoop Distributed File System is no menial task. Especially in large-scale systems with many nodes which each require administration.

2.2.2 Managing HDFS with Ambari

Using Ambari to handle the administration of a HDFS cluster can make operations much easier for a human to invoke. However, there is still some knowledge required in order to set it up including: Linux commands and repository additions[14].

However, once a user has overcome any potential difficulties Ambari does make the process of setting up and managing a cluster far easier by the use of a web GUI. The interface guides a user with lots of explanations on how their actions will affect the cluster.

Although this method has been shown by Hadoop developers to reduce the time it takes to manage errors within the cluster and to get an initial set up running, our proposed project wishes to take it a step further by removing the need for user intervention thoroughly. The Ambari web GUI still requires some application knowledge to be fully utilised, including a full understanding of the Hadoop ecosystem and its packages. Our system wishes to allow a user to simply run it and begin using its functionality rather than having to learn the stack first.

2.2.3 Managing GPFS

In order to determine how difficult GPFS is to manage and administrate we must look into the product's documentation. GPFS's administration and programming reference itself is over 500 pages long and states the following about who should be reading it:

This information is designed for system administrators and programmers of GPFS systems. To use this information, you should be familiar with the GPFS licensed product and the AIX, Linux, or Windows operating system, or all of them, depending on which operating systems are in use at your installation.

Where necessary, some background information relating to AIX, Linux, or Windows is provided. More commonly, you are referred to the appropriate documentation.[20]

2.2.4 Summary

In this section we have discussed the amount of user intervention required in some existing distributed file systems. Using this information we can infer how our system should operate in terms of administration and management. This will allow us to produce requirements matching our theme of removing user intervention as adequately as we can from the system, allowing it to run and self-configure autonomously. As we have shown some distributed systems have far more configuration than others, making their operation difficult and the time frame to have a fully implemented system longer. These are features which our project plans to avoid.

2.3 Determining Node Fitness

The proposed system intends to use some form of weighting algorithm in order to determine which data nodes should be used to store which files. If the replication amount is less than the number of nodes available for storage, considerations

must be made for which node is optimal. In order to understand this idea, we must explore methods which have already been implemented in commercial distributed file systems.

2.3.1 File Placement in HDFS

The Hadoop file system's approach to determining file replication placement is found within its source code inside the **ReplicationTargetChooser** class and is performed using its member function **chooseTarget()**. The comments surrounding this code state that placement is dependant initially on randomisation, having the first replication be a random data node from one rack. If the replication value is greater than one, subsequent nodes are chosen based on rack location. The second file location is placed on a random data node on a separate rack to the first data node. Any further replications required are randomly chosen as data nodes on the same rack as the data node chosen second.[2]

From this, it is clear to see that data locations are not chosen based on environmental factors (unlike what we intend for the intended system). There are both advantages and disadvantages because of this. For example, using pure randomisation does mean that less processing power is required for data to be distributed and HDFS's method of distribution does allow copies to be far enough away from each other to increase data redundancy and reliability.

However, this method creates an issue where certain data nodes can be overloaded with files in comparison to others as no memory usage is taken into account. This can also potentially lead to more disk failures on a large system which handles many files, as certain data nodes may require exponentially more read and write operations on the disk, reducing its lifespan.

2.3.2 File Placement in GFS

Google's distributed file system takes a slightly more efficient approach to file distribution than HDFS. A GFS cluster consists of a single master node and sev-

eral data nodes (named Chunkservers). Files are broken down into fixed chunks whose size are defined by the user on set up. Once a file has been broken down into chunks it is given a 64-bit label in order to maintain the logical mapping of the file. Chunks are then evenly distributed throughout the cluster to ensure that no single Chunksever dwarfs another in chunks stored.[6]

In comparison to HDFS, one can see that GFS avoids potential disk failure due to overuse on data nodes by requiring an even distribution of chunks. In a large system disk failures are bound to happen, taking these measures to avoid this can reduce these errors from occurring.

Although Google's method allows for an even distribution of files it is not necessarily the optimum method of storing files across the cluster, as once again (similarly to HDFS), factors such as the number of chunks stored on individual nodes is not taken into account.

2.3.3 File Placement in IBM's GPFS

File placement within IBM's clustered file system is determined using 'File Policies' which are user-defined. They can be used to automatically place newly created files in specific storage pools. These pools are collections of disks which are used to create a system similar to a data node as seen in HDFS and GFS.[10]

File policies are written by users in an SQL style language and contain several rules which specify conditions. When a rule condition is met GPFS can apply the rule to determine the end location of the file. GPFS contains many conditions which can be used to apply rules, such as:

- Current date and time
- Date and time the given file was last accessed
- Date and time where the file was last modified
- File name

- File extension
- File size

IBM's policies allow their file system to perform the performance optimisation analysis which is intended to be implemented in the proposed system. However, these policies require much domain knowledge and user intervention in order to successfully implement. The proposed system will avoid this issue by having the software automatically determine the best location in which to store files instead of checking user-defined rules. There is also an issue of processing time as if someone defines a rule which is particularly long or complex, much processing power could be required in order to successfully fulfil the rule[10].

2.3.4 Potential Methods for Node Choices

Below we will be discussing three different methods which the proposed system could use in order to determine which nodes files are sent to. We will explore ant colony optimisation, genetic algorithms and discuss the design of a basic node scoring system based on collected metrics.

2.3.4.1 Ant Colony Optimisation

ACO is a method of solving issues regarding combinatorial optimisation problems[15]. It works to model the actions of ants in an ant hill. When ants determine a best route somewhere they leave a pheromone trailer for other ants, which also use pheromones for communication.

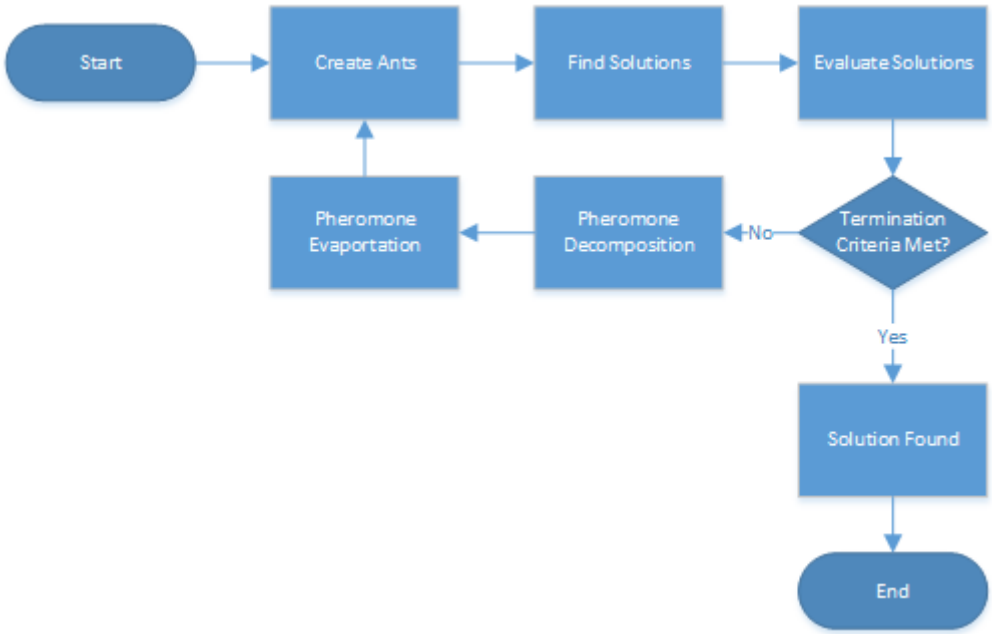
The first ACO algorithm proposed was named 'The Ant System' and was used to figure out a 'Travelling Salesman' problem using 75 cities[16]. It is important to note that the ACO algorithm was not intended to solve this one problem, but to be used for the creation of optimised systems[15]. It managed to compute with the same speed as evolutionary algorithms however did not manage to solve the problem quicker than algorithms which had been developed to specifically solve the problem[16].

These trails are stored as distributed, numerical information which the master node will adapt during the algorithm's execution to find nodes which seem to be the best current locations for files. The 'ants' in this sense are looking for the best route to take based on meta data that the data nodes will supply the master node with (see requirements section for more details).

This method could be promising, as nodes are constantly being checked for their fitness and an appropriate node is always ready when a file system modification is requested. The only problems being that the optimisation algorithm will have to run constantly, which could cause processing problems once the cluster has been scaled up. When more nodes are added, routes become far more complicated to compute.

Below is a basic flow diagram outlining the operations of an Ant Colony Optimisation algorithm:

Figure 2.1: Ant-Colony Optimisation Flow



2.3.4.2 Genetic Algorithms

Another potential method for determining most appropriate location to store files is through the use of genetic algorithms. They were invented by John Holland at the University of Michigan in the 1960s to study the phenomenon of nature's ability to adapt to its surroundings and to translate this into working computer systems[17].

Genetic Algorithms are considered to be adaptive heuristic (enabling a person to discover or learn something for themselves) search methods which utilise randomisation whilst exploiting historical information to find best performance within the search criteria. In layman's terms, these algorithms follow the principles of Charles Darwin's 'survival of the fittest'[17].

The easiest way to express what occurs in a genetic algorithm is in Mitchell's publication on the subject. She defines a basic genetic algorithm as follows:

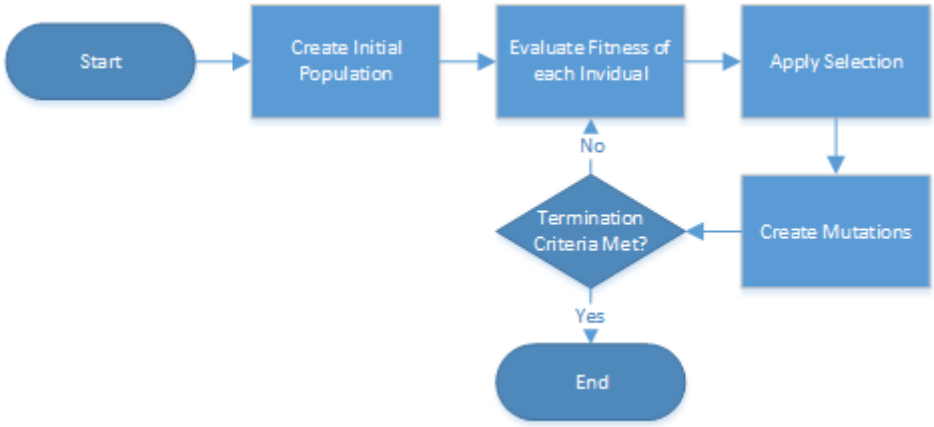
- Individuals in a population compete for resources and mates.
- Those individuals most successful in each generation will produce more offspring than those that perform poorly.
- Genes from 'good' individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
- Thus each successive generation will become more suited to their environment.

This method could provide value when selecting nodes as the best performing nodes will be deduced by the algorithm and subsequently files can be sent to it. This, similarly to ACO, raises an issue again of scalability, as commodity hardware may not be the best fit for a potentially high amount of processing. In modern day, most genetic systems we see tend to use quite high-end hardware to produce solutions. GA's dependency on randomisation can also raise an issue

in that it is uncertain how many populations and runs are required in order to produce a result that could be considered optimum[19].

Below is a flow diagram outlining the basic operation of a genetic algorithm:

Figure 2.2: Genetic Algorithm Flow



2.3.4.3 Basic Scoring Methods

A third option could be to produce a completely unique scoring method for the proposed system. This has a few advantages and disadvantages to the previously discussed methods. Namely, the previous algorithms are evolutionary and will require multiple iterations until generating an answer we can consider 'optimum'[19]. Having a custom scoring algorithm means that the results will always be optimal (providing the developed algorithm is correct).

However, the discussed methods are very tested and have shown to produce very optimal results. If a custom algorithm's designer does not fully understand the domain a produced algorithm may not produce optimal results. This raises a trade, do we take garmented optimisation over more processing and time, or instead take lower computation times with a potential of poorer optimisation.

For this reason it is appropriate that all methods are produced and compared in order to evaluate which produces the best results for this system.

2.3.5 Summary

In this section we have discussed different methods for determining file location based on existing distributed systems and our own algorithm design. From this we can produce requirements for each different algorithm and compare them appropriately to find one which suits our purpose best and provides the greatest amount of efficiency for processor use. As it is intended for these systems to run on a small Raspberry Pi board it is important that whichever algorithm is ultimately chosen it takes into account the smaller amount of processing power available on these boards. Through adequate benchmarking and testing it should be possible to establish and implement a best fit algorithm.

2.4 Determining File Replication

This section will discuss among the three discussed distributed file system how each handles file replication. More specifically how each different system determines how many copies of a file should be placed within the cluster. All three chosen file systems utilise this functionality differently, with each having a different emphasis on randomisation, automation and customisation.

2.4.1 HDFS

The Hadoop Distributed File system utilises replication in a way that is probably the most simple, but least customisable method out of the three distributed systems discussed. HDFS simply uses an XML configuration file in which the user assigns an integer value to the replication property.[2]

Although this method works effectively when used alongside HDFS's node target chooser it does not allow for a large amount of customisation or applying rules to files for changing their replication based on certain factors. Other

Hadoop packages may fix this issue however on its own, HDFS will replicate any given file the number of times stated in the configuration file.

This means that, by default, no aspects of the files being added or their destination environment is taken into account, potentially leading to inefficient replication of file data.

2.4.2 GFS

Of the three DFS systems that have been discussed, Googles GFS is the only one which implements some form of automation for files added to the file system. GFS sets a minimum replication value of three, meaning that regardless of file size or type there will always be at least three available copies within the system. GFS will then increase the replication value based on how often the file is required, whether or not it has been marked as a 'mission-critical' or its size, with larger files generally having more replications.[6]

This is the type of replication automation which the proposed system aims to provide. When combined with an adequate method of node selection for file placement, this system will ensure that both the redundancy and availability of data are at maximum. The only issue being that all of the file-access data must be stored on disk in meta data files, potentially using space that could be utilised by the DFS.

2.4.3 GPFS

IBM's General Parallel File System's replication process is entirely created by the administrator when a cluster is being created. Replication is achieved by modifying configuration variables on specific file systems and has no built-in method of determining appropriate redundancy.

Not only does the user have to specify their replication amount, but any existing files on the system once replication values have been changed must be

manually replicated using command line instruction.

2.4.4 Summary

In this section we have discussed different methods for choosing how many times a file should be replicated throughout a file system. Based on existing systems it seems that this is entirely down to user configuration and has not been implemented using some form of learning algorithm. This means that we can produce an algorithm to take care of this task and compare it to the efficiency of default replication values of the existing systems and hopefully find out which is more efficient.

Chapter 3

Requirements

3.1 Requirement Specification

Each requirement is priorities using MoSCoW prioritisation which uses the following letters and meanings:

- 'M' - Must Have: These requirements are necessary for the project to be fully realised.
- 'S' - Should Have: These requirements should be implemented but are not necessary for project realisation.
- 'C' - Could Have: These requirements are time restricted and should only be implemented time-scale permitting.
- 'W' - Wont Have: These requirements will not be implemented. Either due to redundancy or project time frame.

3.2 Non-Functional Requirements

Below are all defined non-functional requirements for the proposed system. These specify a criteria used for judging the operation of software we will produce.

3.2.1 Usability

Usability is the degree to which a software can be utilised by users to achieve or surpass objectives which are considered quantifiable (i.e measurable, for example over time). This section will identify the usability non-functional requirements we intend to meet within our project's development.

Table 3.1: Usability NFRs

#	Description	Priority	Rationale
NFR-U-1	It shall be possible to accomplish any task within the cluster using just a keyboard.	M	Most systems like this function using API calls. Because of these, a command-line interface is required for operations to take place manually.
NFR-U-2	All delays in the system with be annotated, with text, in-console displaying progress.	S	The system must inform the user regularly to prevent user-error and to debug efficiently.

3.2.2 Performance

Performance requirements are characterised by the amount of performable tasks on a computer system compared to the time and resources used. In this section, we will discuss the performance objectives of the system and what the requirements are.

Table 3.2: Performance NFRs

#	Description	Priority	Rationale
NFR-P-1	A new data node should be added to the cluster within 5 seconds.	M	Nodes will update the master node every five seconds. Therefore 5 seconds should be enough time for a node to update the master accordingly.
NFR-P-2	A file of approximately 1MB should be added to the cluster within 5 seconds.	M	Hadoop currently holds up as having a throughput of 85MB/sec. In our case, five seconds is enough time for a file of this size to be sent.
NFR-P-3	A file of approximately 1MB should be renamed in the cluster within 5 seconds.	M	See above.
NFR-P-4	A file of approximately 1MB should be removed from the cluster within 5 seconds.	M	See above.
NFR-P-5	A directory should be added to the cluster within 5 seconds.	M	See above.
NFR-P-6	A directory should be removed from the cluster within 5 seconds.	M	See above.

3.2.3 Operational

These requirements refer to the system's ability to remain in a reliable state of operation based on numerous factors. In this section we will discuss what the operational requirements of our intended system are.

Table 3.3: Operational NFRs

#	Description	Priority	Rationale
NFR-O-1	The system must be able to run 24 hours a day 7 days a week.	M	Distributed file systems may be required to perform operations at any time based on the request's size.
NFR-O-2	Nodes should be able to communicate using both a wireless and Ethernet connection.	C	This feature is easy to accommodate for as most programming languages do not require a specification of what internet adaptor to use.
NFR-O-3	The system must be able to run on a Raspberry Pi board.	M	These boards provide a simple interface and will ensure the product is compatible with multiple operating systems.

3.2.4 Maintainability and Portability

These requirements express the ease with which the project can be maintained in order to isolate, correct or repair defects and to maximise a product's lifespan. In the table following we list the maintainability and portability requirements our system aims to implement.

Table 3.4: Maintainability and Portability NFRs

#	Description	Priority	Rationale
NFR-MP-1	The system must be able to run on Linux-based and Windows-based systems.	M	Allowing users to create a system using multiple operating systems will increase ease of use.
NFR-MP-2	The system shall provide a set-up which has minimum human intervention.	M	The aim of our project is to move configuration towards automation and so the user will not be required to do much.
NFR-MP-3	The system shall provide administration which has minimum human intervention.	S	See above.

3.2.5 Security

Table 3.5: Security NFRs

#	Description	Priority	Rationale
NFR-SC-1	All nodes require a user name and password to access.	C	Ensuring that files remain secure is an important facet of any file system.
NFR-SC-2	Any node-specific information transferred will be encrypted.	C	Ensuring that node activity remains secure is important in preventing theft of data.

3.3 Functional Requirements

Below are all defined functional requirements for the proposed system, they have been separated into relevant sections. These define functions of the software we will develop and its components. They have been defined using the Volere template in order to fully describe the functionality of our system.

3.3.1 File Operations

Table 3.6: File Operations FRs

#	Description	Priority	Rationale
FR-FO-1	A file can be added to the cluster within the CLI	M	Adding files is basic expected functionality of a distributed file system.
FR-FO-2	A file can be renamed in cluster within the CLI.	M	See above.
FR-FO-3	A file can be removed from the cluster within the CLI.	M	See above.
FR-FO-4	A directory can be added to the cluster within the CLI.	M	See above.
FR-FO-5	A directory can be renamed in cluster within the CLI.	M	See above.
FR-FO-6	A directory can be removed from the cluster within the CLI.	M	See above.

3.3.2 Resource Management

Table 3.7: Resource Management FRs

#	Description	Priority	Rationale
FR-RM-1	A new data node can be added to the cluster by providing it with the master node's IP address.	M	In order to fulfil the idea of minimal configuration, a user should only need the master's location in order to implement the system.
FR-RM-2	New replication values must be calculated every time a new data node is added.	M	So that the system can remain efficient, recalculations for efficiency should take place once the cluster scales.
FR-RM-3	Data nodes shall send a packet every 5 seconds to the master node containing their status information	M	So that the node selection algorithms make the best decisions, the master must be regularly updated with node information.
FR-RM-4	Most fit data nodes must be selected through a genetic algorithm	M	So that we can fully compare the power of each algorithm, each one must be implemented and benchmarked.
FR-RM-5	Most fit data nodes must be selected through a custom ranking algorithm.	M	See above.
FR-RM-6	Most fit data nodes must be selected through ant colony optimisation	M	See above.

3.4 Requirements Testing

In order to test that each requirement has been fulfilled we will be using acceptance tests to compare the completed system with the detailed requirements. Although the requirements may change throughout development and implementation this section will be kept up-to-date.

Each acceptance test with be completed using the command-line as sole interface of the application. Each test will be documented in a way which indicates its test number, which requirement or requirements it will evaluate, any pre-requisites required for the test and any expected outcomes. These results will be compiled into a table containing whether or not the test passed or failed, including any comments made about the test.

3.4.1 Acceptance Test List

Table 3.8: File Operations Acceptance Tests

1. File Operations		
Requirements Covered:	FR-FO-1 to FR-FO-3	
Prerequisites:	• The cluster is running with a master node and several data nodes. • A test file named 'test.txt' is available.	
Steps	Instructions	Expected Outcome
1	Type the command 'addfile test.txt test.txt' into the master node	The software should add the file to the cluster.
2	Type the command 'deletefile test.txt' into the master node	The software should delete the file from the cluster.
3	Type the command 'renamefile test.txt test2.txt' into the master node	The software should rename the file within the cluster.

Table 3.9: Directory Operations Acceptance Tests

2. Directory Operations		
Requirements Covered:	FR-FO-4 to FR-FO-6	
Prerequisites:	• The cluster is running with a master node and several data nodes.	
Steps	Instructions	Expected Outcome
1	Type the command 'adddir test/' into the master node	The software should add the directory to the cluster.
2	Type the command 'deletedir test/' into the master node	The software should delete the directory from the cluster.
3	Type the command 'renamedir test/ test2/' into the master node	The software should rename the directory within the cluster.

Table 3.10: Resource Management Acceptance Tests

3. Resource Management		
Requirements Covered:	FR-RM-1 to FR-RM-3	
Prerequisites:	• The cluster is running with just the master node.	
Steps	Instructions	Expected Outcome
1	Type the command 'connect 192.168.0.2' into the data node.	The software should add the data node to the cluster. This can be checked on the master node using the 'nodes' command.
2	Type the command 'nodes' into the master node.	The software should return a list of all connected data nodes and their metrics.

Table 3.11: Node Choice Algorithms Acceptance Tests

4. Node Choice Algorithms		
Requirements Covered:	FR-RM-4 to FR-RM-6	
Prerequisites:	• The cluster is running with several data nodes and a master node.	
Steps	Instructions	Expected Outcome
1	Type the command 'use GA' into the master node	The software should start using the genetic algorithm to perform its node choices. Add a file to the cluster.
2	Type the command 'use ACO' into the master node	The software should start using ant-colony optimisation to perform its node choices. Add a file to the cluster.
3	Type the command 'use CUSTOM' into the master node	The software should start using the custom node selection algorithm to perform its node choices. Add a file to the cluster.

Chapter 4

Design

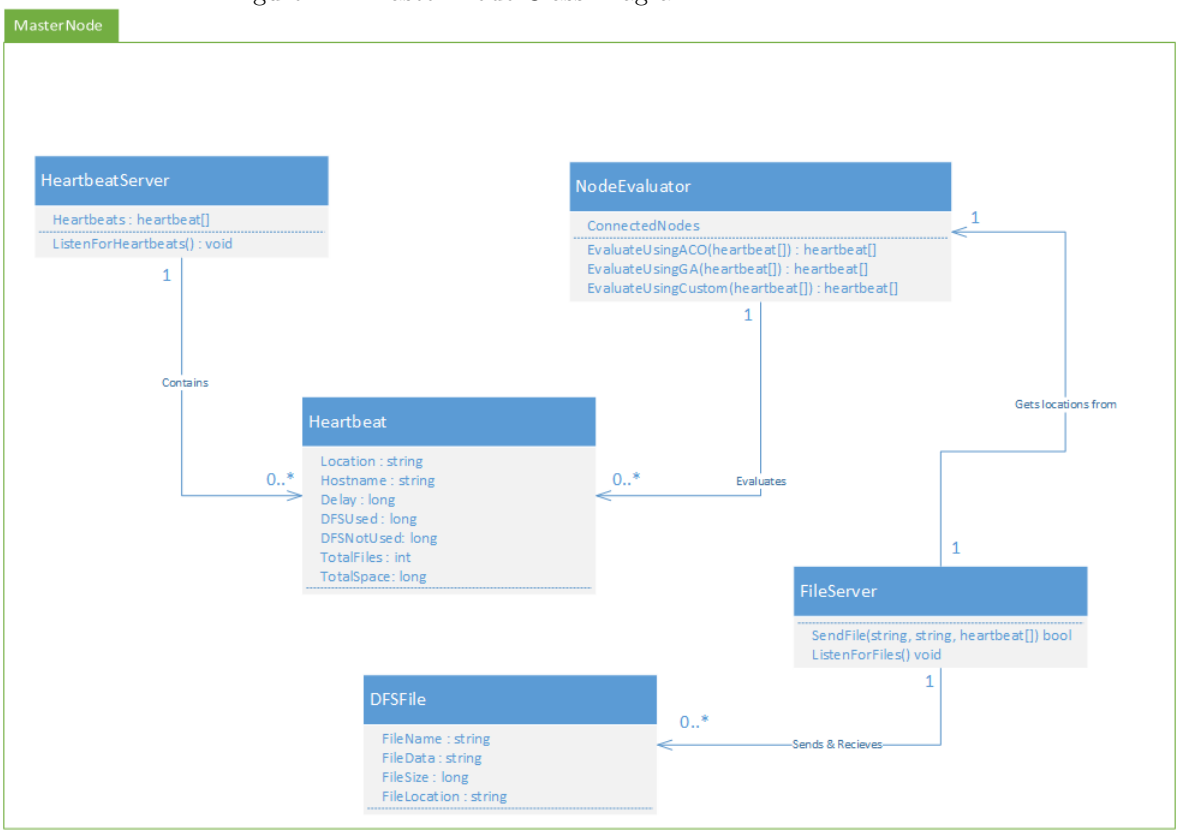
4.1 High-Level Design

The design of our distributed file system consists of two key program boundaries. These are the master and slave node systems. These will be modelled using UML class and sequence diagrams. As the GUI will be command-line there will be no GUI design shown here.

4.1.1 Master Node High-Level Class Diagram

Below is the basic high level class diagram for the intended system's master node functionality. It consists of a server to interpret data node heartbeats, an evaluator to make file decisions and a file server in order to transmit files across the data nodes.

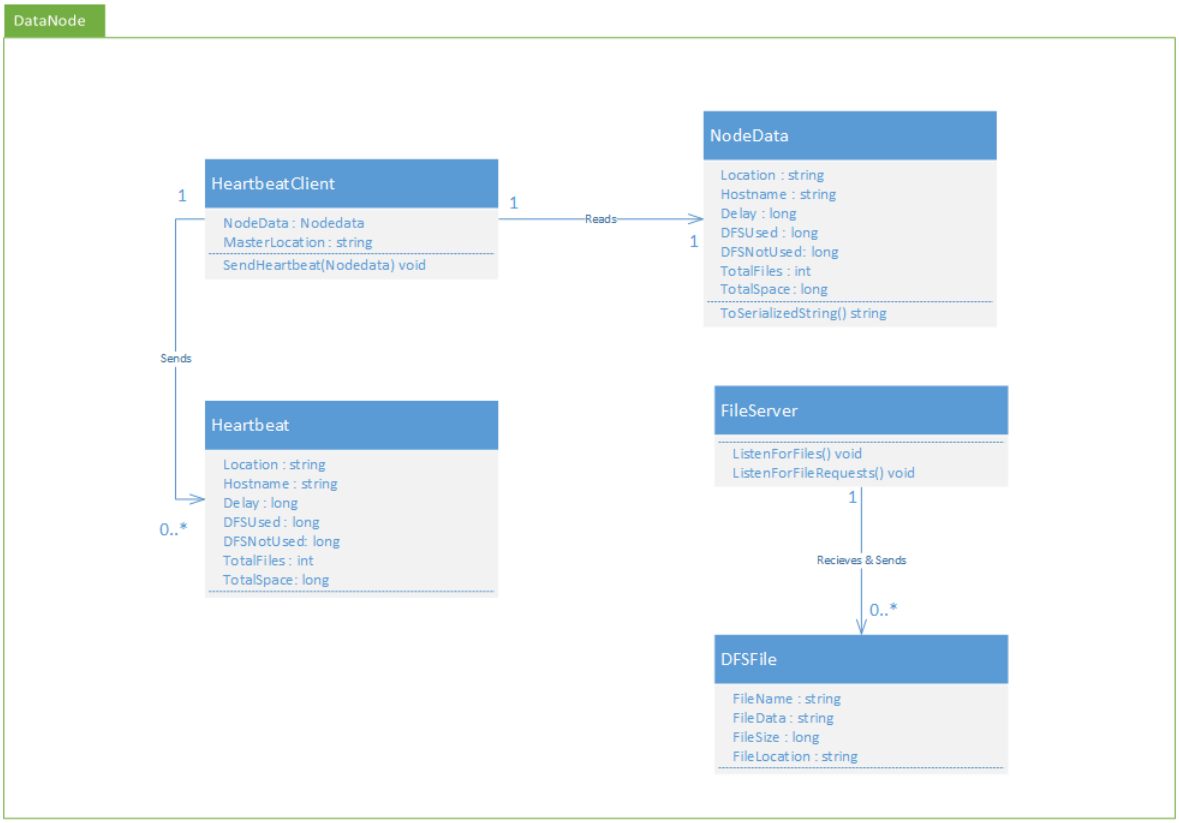
Figure 4.1: Master Node Class Diagram



4.1.2 Data Node High-Level Class Diagram

Below is a high level description of the classes involved in the data node's functionality. It utilises a client in order to send heartbeats regularly to the master node and a file server which will receive and send files based on commands from the master node.

Figure 4.2: Data Node Class Diagram



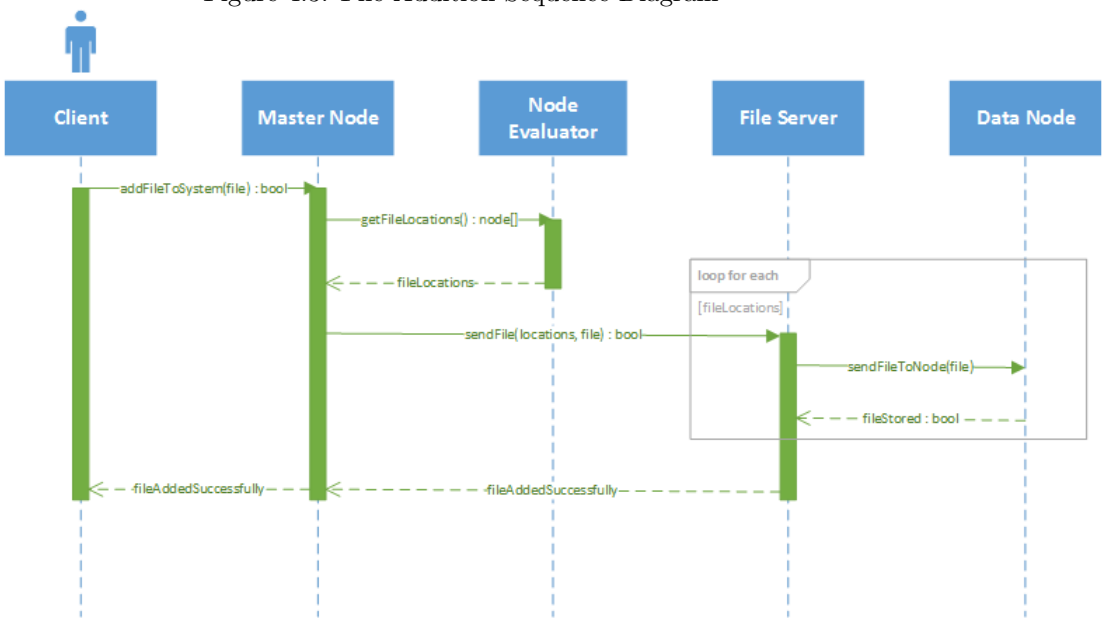
4.2 Sequence Diagrams

A sequence diagram is used in order to display the activity between objects in a sequential order. Their primary use is in the transition for requirements (shown as use-cases) to become a more formal level of refinement. The use cases shown previously (See section: Requirements) can be broken down into several sequence diagrams.

4.2.1 Adding a File

Below is the diagram for adding a file to the distributed file system. It details the interaction between both client and master node including the subsequent operations required to add the file to our proposed file system. It also details the iterative nature of the node evaluation code which utilises redundancy to ensure data availability.

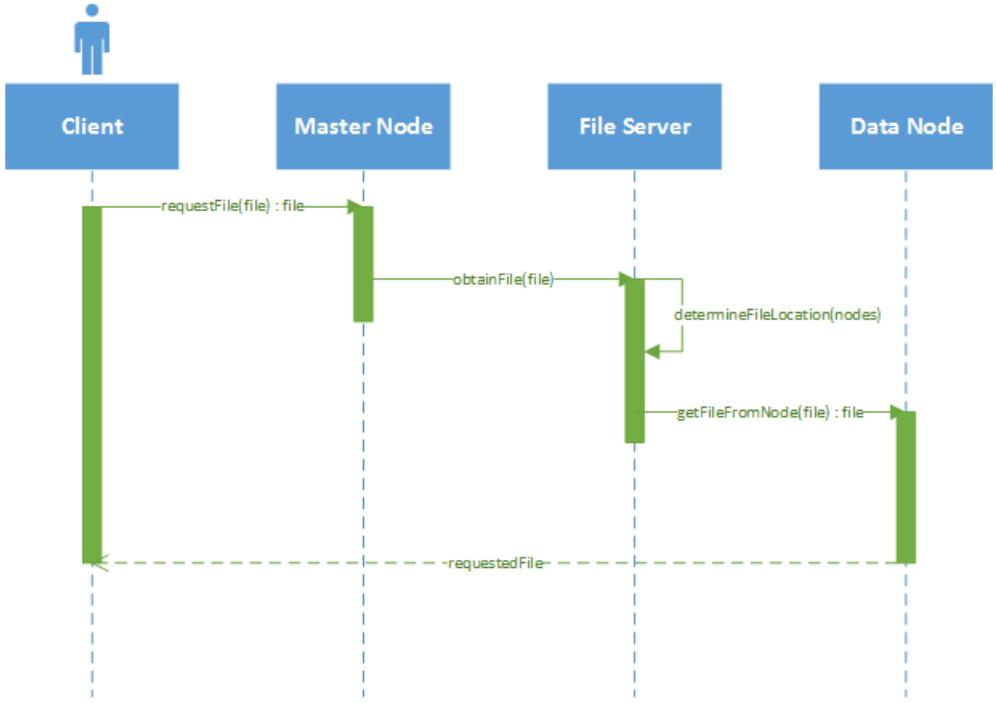
Figure 4.3: File Addition Sequence Diagram



4.2.2 Retrieving a File

Below is the diagram for retrieving a file from the distributed file system. It details the interaction between both client and master node including the subsequent operations required to obtain the file from our proposed file system.

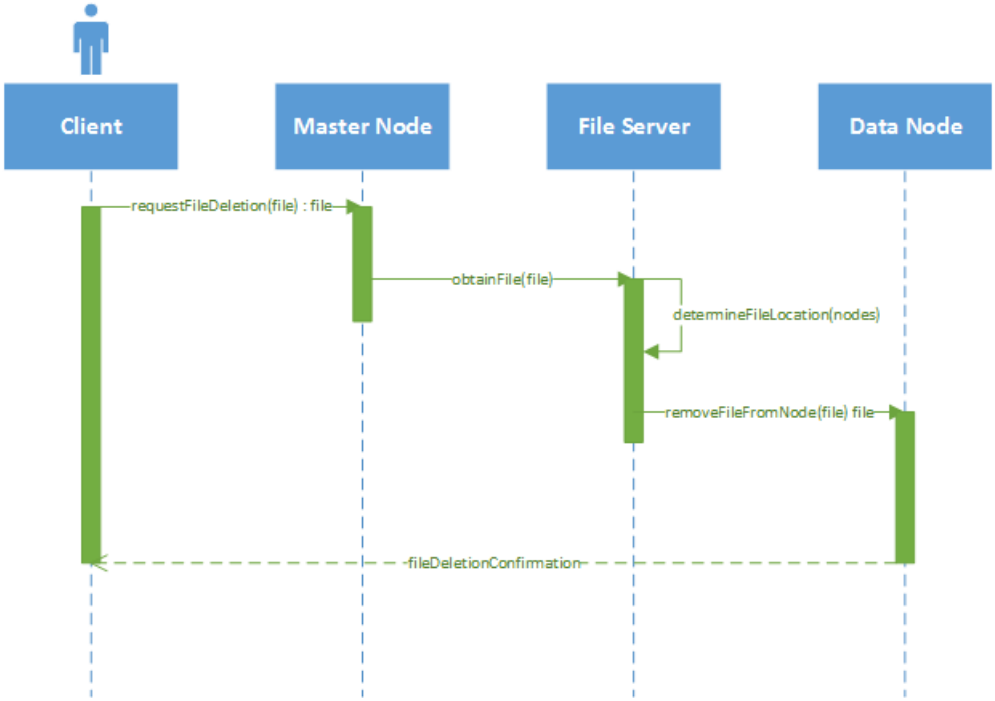
Figure 4.4: File Retrieval Sequence Diagram



4.2.3 Removing a File

Below is the diagram for removing a file from the distributed file system. It details the interaction between both client and master node including the subsequent operations required to delete the file from our proposed file system. Upon receiving a deletion request the master node will look up which nodes contain the requested file and deletes it accordingly.

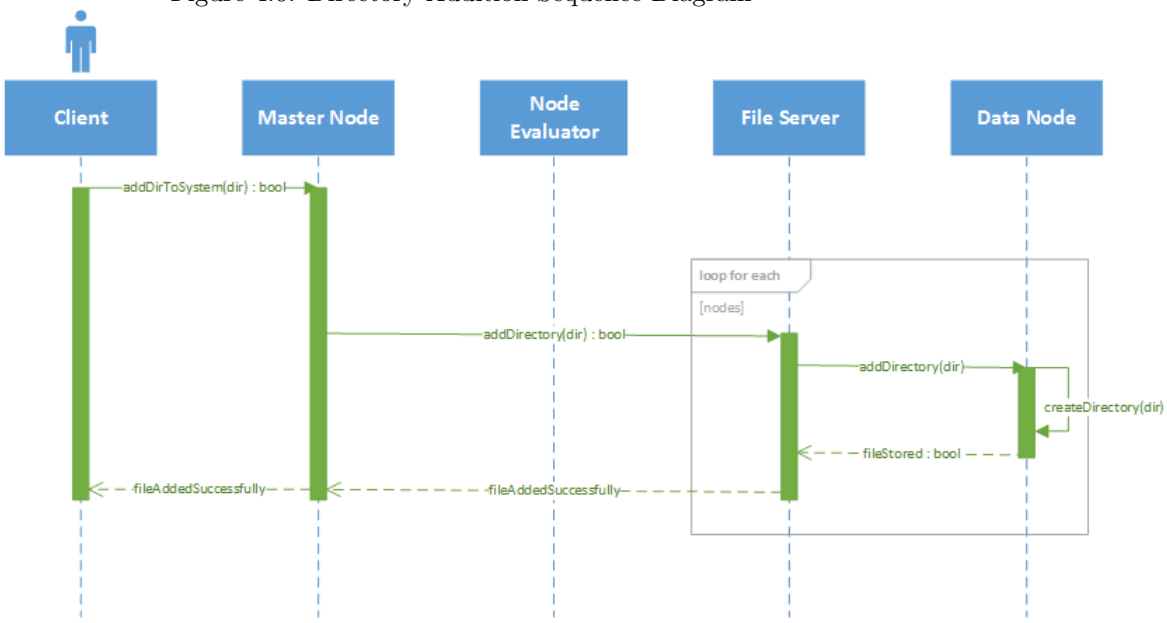
Figure 4.5: File Removal Sequence Diagram



4.2.4 Adding a Directory

Below is the diagram for adding a directory to the distributed file system. It details the interaction between both client and master node including the subsequent operations required to add the directory to our proposed file system. Upon receiving a folder addition request the master node will create the directory on all nodes in the system.

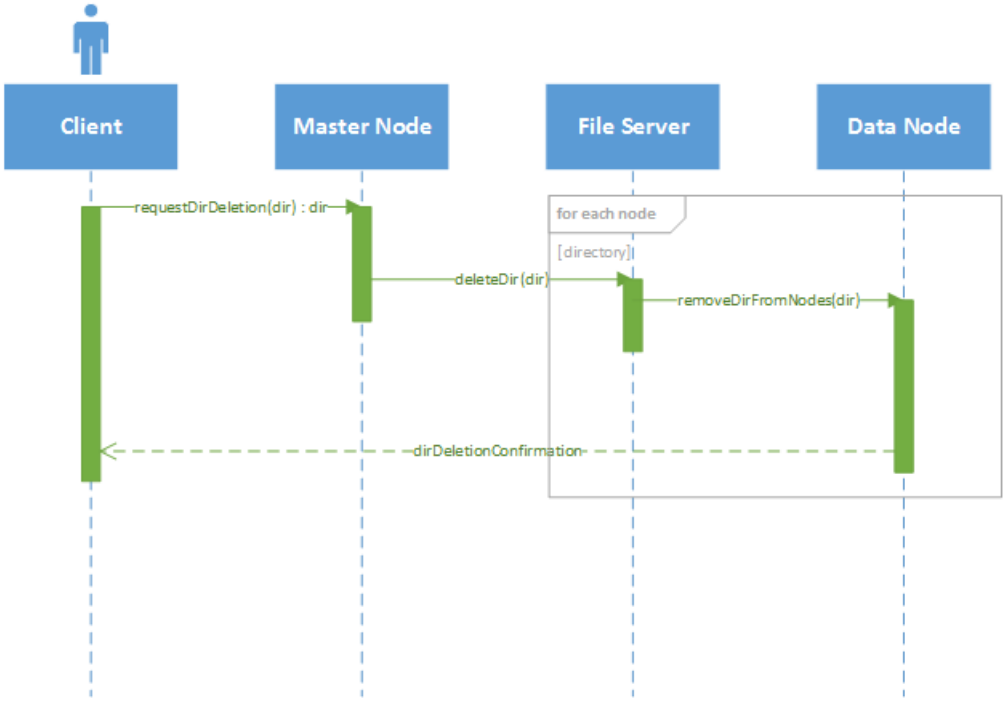
Figure 4.6: Directory Addition Sequence Diagram



4.2.5 Removing a Directory

Below is the diagram for removing a directory from the distributed file system. It details the interaction between both client and master node including the subsequent operations required to delete the directory from our proposed file system. Upon receiving a folder deletion request the master node will look up which nodes contain that directory and delete it accordingly.

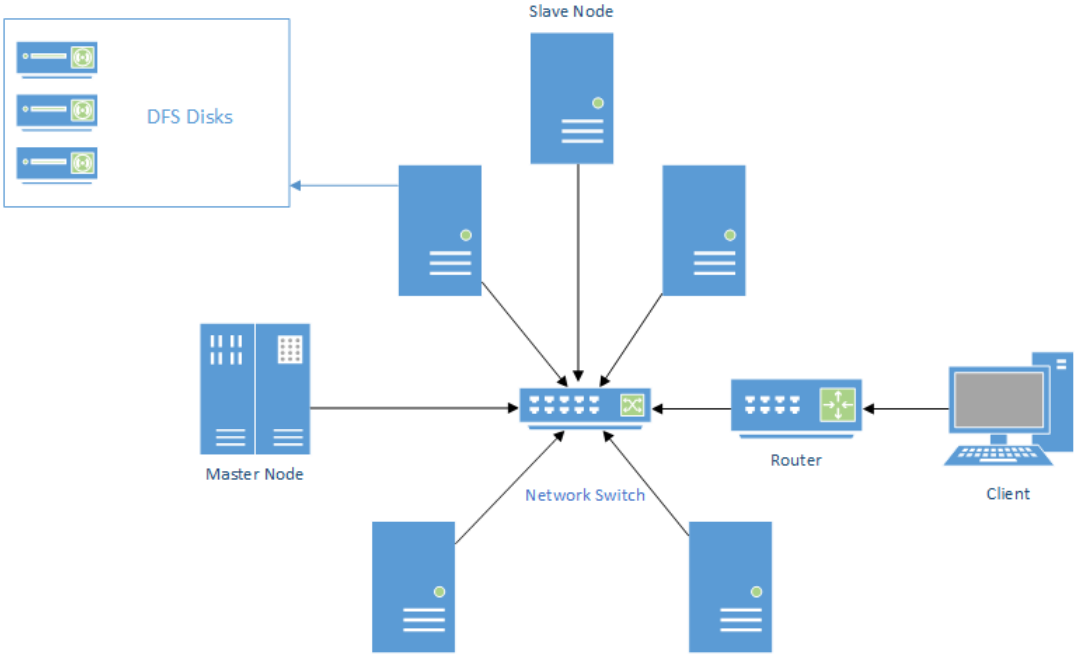
Figure 4.7: Directory Removal Sequence Diagram



4.3 Network Architecture

This diagram illustrates the typical network architecture of the proposed system. It uses a star-like topology where all nodes are connected to a switch. The client then connects to the master node which performs operations on the slave nodes. Each slave node can consist of multiple disks for storage.

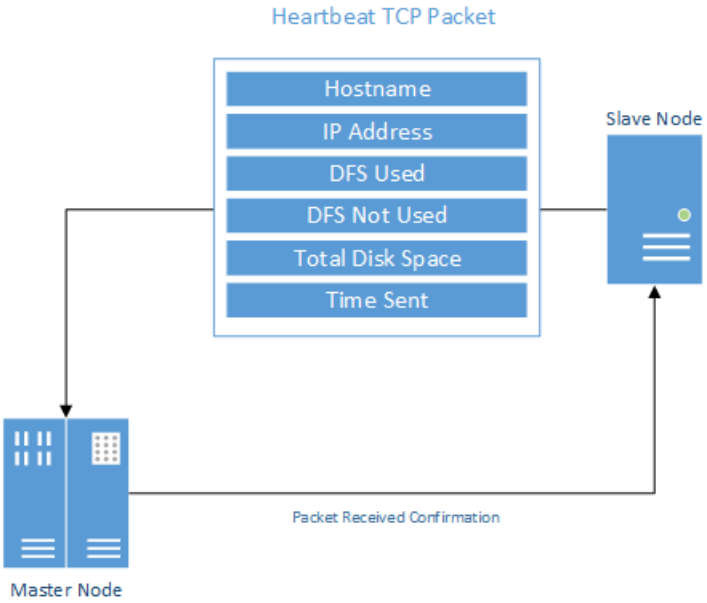
Figure 4.8: Network Architecture Diagram



4.3.1 Heartbeat Communication

Finally, this diagram illustrates the communication between the master and data nodes which occurs approximately every five seconds. The data node will update the master node with its current evaluable information. This allows the node to make accurate assumptions as to which data node to use for storing data at the current time.

Figure 4.9: Heartbeat Communication Diagram



Chapter 5

Implementation

5.1 Implementation Decisions

Prior to implementation, we need to decide several factors of the development stage including IDE and any third-party libraries to use.

5.1.1 IDE

The chosen IDE is called Xamarin Studio and is developed to fully support the Mono library. This means that we can develop and deploy C# applications on cross-platform environments (including Windows and Linux). This means that the developed cluster is fully compatible across multiple operating systems, allowing for nodes to run together on both Windows and Linux environments.

Unfortunately, this means that we cannot take advantage of any other libraries in the C# language other than the System name space. This is due to the fact that Xamarin Studio utilises Mono, an open source, multi-platform version of the .NET framework. This means that anything the system needs to do must be coded from scratch, increasing production time when it comes to implementing ant-colony optimisation and genetic algorithms (as these would normally be implemented using a third-party library). This sacrifice however, allows us to meet the non-functional requirement NFR-MP-1.

5.2 Implementing Node Location Algorithms

In order to choose ideal file locations we must implement an algorithm to determine the best suited nodes as previously discussed. Below we will identify the steps taken in order to implement each of these algorithms in detail.

5.2.1 Custom Ranking Algorithm

The custom ranking algorithm can be explained using mathematical notation. Each node in the cluster can be represented as a set of variables. These are:

- S_i - The total space used by the DFS on a respective node.

- N_i - The total space not used by the DFS on a respective node.
- D_i - The total disk space on a respective node.
- F_i - The total files stored on a respective node.
- L_i - The latency between a node and the master node.

The cluster is also defined by a set of variables. These are:

- n - The total number of nodes in the cluster.
- i - The current node being evaluated.
- Y - The vector containing all nodes.

The algorithm works similarly to how drivers are ranked in a formula one race. For each variable described above, nodes are ranked 1 to n (where n is the total number of nodes in the cluster). For each variable ranking every node's position is added to a total score (shown as y_i) which is used to determine fitness. Below are defined vector positions of each node in the cluster (They are accessed using y_i):

$$Y = (y_1, y_2, y_3, \ldots, y_n) \tag{5.1}$$

Each vector is affected using the following algorithm to rank nodes:

$$y_i = (rank(S_i) + rank(N_i) + rank(D_i) + rank(F_i) + rank(L_i)) \tag{5.2}$$

Once the ranking is completed the top x number of nodes are chosen (where x is the current replication value for that file and Z is the vector containing chosen 'fit' nodes) to receive the file. This is achieved using a 'top' function such as:

$$Z = top(x, Y) \tag{5.3}$$

In code, this equates to small functions which rank the nodes based on their different attributes using linq (lambda expressions). Below is an example of one ranking function in C# defined within listing 5.1.

Listing 5.1: Evaluate Nodes by Latency

```
1  private List<KeyValuePair<Heartbeat, int>> ScoreNodesByLatency (List<Heartbeat>
       heartbeats)
2  {
3    // Use a keyvalue pair to ensure no node is evaluated twice in one go.
4    var retVal = new List<KeyValuePair<Heartbeat, int>> ();
5
6    // Order the heartbeats by their latency.
7    var beatsByLatency = heartbeats.OrderByDescending (x =>
         x.Delay.TotalMilliseconds);
8
9    // For each heartbeat.
10   for (int i = 0; i < beatsByLatency.Count(); i++) {
11     // Add heartbeat and score to the list.
12     retVal.Add (new KeyValuePair<Heartbeat, int> (beatsByLatency.ElementAt (i),
           i));
13   }
14
15   // Once complete, return ordered descending by score.
16   return retVal.OrderByDescending(x => x.Value).ToList();
17 }
```

Each other ranking function behaves similarly and all scores are combined in the following function described by listing 2.

Listing 5.2: Obtain Best Fit Nodes

```
1  public List<KeyValuePair<Heartbeat, int>> GetBestNodes (int replication,
       List<Heartbeat> heartbeats)
2  {
3    // Container for total scores.
4    var totalScores = new List<KeyValuePair<Heartbeat, int>> ();
5    // Get scores based on properties.
6    var latencyScores = this.ScoreNodesByLatency (heartbeats);
7    var dfsNotUsedScores = this.ScoreNodesByDfsNotUsed (heartbeats);
8    var dfsUsedScores = this.ScoreNodesByDfsUsed (heartbeats);
9    var totalFilesScores = this.ScoreNodesByTotalFiles (heartbeats);
10   var totalSpaceScores = this.ScoreNodesByTotalSpace (heartbeats);
11   // Loop through each node.
12   foreach (Heartbeat beat in heartbeats) {
13     // Reset the score.
14     int currentScore = 0;
15
16     // Get relevant score and add it to the total.
17     currentScore += latencyScores.Where (x => x.Key.Source.Equals (beat.Source))
18                                  .FirstOrDefault ().Value;
19     currentScore += dfsNotUsedScores.Where (x => x.Key.Source.Equals
           (beat.Source))
20                                  .FirstOrDefault ().Value;
21     currentScore += dfsUsedScores.Where (x => x.Key.Source.Equals (beat.Source))
22                                  .FirstOrDefault ().Value;
23     currentScore += totalFilesScores.Where (x => x.Key.Source.Equals
           (beat.Source))
24                                  .FirstOrDefault ().Value;
25     currentScore += totalSpaceScores.Where (x => x.Key.Source.Equals
           (beat.Source))
26                                  .FirstOrDefault ().Value;
27     // Add this total value to the list with its heartbeat.
28     totalScores.Add (new KeyValuePair<Heartbeat, int> (beat, currentScore));
29   }
30   // Return top 'n' (where n is replication value) nodes ordered descending by
         score.
31   return totalScores.OrderByDescending (x => x.Value).Take (replication).ToList
         ();
32 }
```

5.3 Implementing The File Replication Algorithm

For full data availability replication is required to ensure that in the event of data corruption another copy of the file is readily available. This means that our project must invoke some form of algorithm which can determine how many times a file should be copied across the system. In order to achieve this we analyse file meta data to determine how critical a file is.

This needs to take into account factors such as when the file was created, when it was last accessed, how big the file is and whether or not any flags such as 'Read Only' or encryption have been applied. The theory being that larger files contain more information, files used or created recently pertain to important, current data. That encryption methods connote importance of files and that flags indicate files that have an importance where they should not be modified.

In our code, this is implemented using some basic conditional statements which increment a value, and is implemented in listing 5.3.

Listing 5.3: File Replication Determination

```
public int DetermineFileReplication(string fileSource)
{
  int replication = 1;
  // Get file info.
  FileInfo file = new FileInfo (fileSource);
  // If the file was created in the last 2 months, increase replicaiton.
  if (file.CreationTime > DateTime.Now.AddMonths (-2)) {
    replication++;
  }
  // If the file is greater than 1MB, increase replication.
  if (file.Length > 1000000) {
    replication++;
  }
  // If the file is read only, increase replication.
  if (file.IsReadOnly) {
    replication++;
  }
  // If the file was accessed in the last month, increase replicaiton.
  if (file.LastAccessTime > DateTime.Now.AddMonths (-1)) {
    replication++;
  }
  return replication;
}
```

5.4 Implementing The File Transfer System

So that files can be transferred among nodes a system is required which can bind to data node ports and send files. For this, we use a simple TCP client and server which creates a file stream of a chosen files, separates it into packets of approximately 1KB each and sends them to the appropriate node where it is reassembled and saved.

The file transfer system's code is implemented from listing 5.4.

Listing 5.4: File Transfer Implementation

```
1  public bool SendFile (IPAddress endPoint, string fileLocation, string
       dfsDestination)
2  {
3    bool retVal = true;
4    byte[] sendingBuffer = null;
5    FileStream fileStream = null;
6    TcpClient client = null;
7    NetworkStream netStream = null;
8    int packetCount = 0;
9    int totalLength = 0;
10   int currentPacketLength = 0;
11   // Try to send file, report any exceptions to the user.
12   try {
13     // Use 'using' to auto dispose of streams.
14     using (client = new TcpClient (endPoint.ToString (), FilePort)) {
15       using (netStream = client.GetStream ()) {
16         using (fileStream = new FileStream (fileLocation, FileMode.Open,
               FileAccess.Read)) {
17           // Calculate how many packets we need.
18           packetCount = Convert.ToInt32 (Math.Ceiling (Convert.ToDouble
                 (fileStream.Length) / Convert.ToDouble (BufferSize)));
19           // Loop through the filestream.
20           for (int i = 0; i < fileStream.Length; i++) {
21             // If the file is larger than our buffer.
22             if (totalLength > BufferSize) {
23               // Recaculate packet and total lengths of file.
24               currentPacketLength = BufferSize;
25               totalLength -= currentPacketLength;
26             } else {
27               // Otherwise, adjust total length and write to stream.
28               currentPacketLength = totalLength;
29               sendingBuffer = new byte[currentPacketLength];
30               fileStream.Read (sendingBuffer, 0, currentPacketLength);
31               netStream.Write (sendingBuffer, 0, sendingBuffer.Length);
32             }
33           }
34         }
35       }
36     }
37   } catch (Exception ex) {
38     Console.WriteLine (ex.ToString ());
39     retVal = false;
40   }
41   return retVal;
42 }
```

The server side of the file transfer system requires a constant check on the given port. For this we have assigned port 13001 as the file transfer port. The server constantly checks whether a file is waiting for transfer and assembles it from the TCP stream placing it in the correct directory. Its implementation is demonstrated in listing 5.5:

Listing 5.5: File Reception Implementation

```
1  private void ListenForFiles ()
2  {
3    // Start listening for files.
4    this.FileListener.Start ();
5
6    byte[] recData = new byte[BufferSize];
7    int recBytes = 0;
8
9    // Loop forever.
10   while (true) {
11     // Catch and report errors but continue.
12     try {
13       TcpClient currentClient = null;
14       NetworkStream netStream = null;
15       FileStream fileStream = null;
16       int totalRecBytes = 0;
17
18       // If a file is waiting.
19       if (this.FileListener.Pending ()) {
20         // Use 'using' to auto dispose streams.
21         using (currentClient = this.FileListener.AcceptTcpClient ()) {
22           using (netStream = currentClient.GetStream ()) {
23           // Get stream info and create file.
24             using (fileStream = new FileStream ("test.txt",
                       FileMode.OpenOrCreate, FileAccess.Write)) {
25               // Loop through file and write to DFS.
26               while ((recBytes = netStream.Read (recData, 0, recData.Length)) >
                       0) {
27               fileStream.Write (recData, 0, recBytes);
28               totalRecBytes += recBytes;
29               }
30             }
31           }
32         }
33       }
34     } catch (Exception ex) {
35       // Log any errors.
36       this.Log.CreateLog (ex);
37     }
38   }
39 }
```

5.5 Implementing The Heartbeat System

In order for our ranking algorithms to be fully effective we need a method for the master node to obtain information about all nodes within the cluster. This has been implemented using serialisation and TCP/IP communication.

The simplest implementation is to create a server on the master node which responds to a certain port. For ease, the port used for sending and receiving node information is 13000. On top of this, each data node requires a TCP client which will serialise and transfer their node information which is parsed by the master upon reception. The heartbeat server is implemented in listing 5.6:

Listing 5.6: Heartbeat Listener Implementation

```
public void ListenForHeartbeats ()
{
  Console.WriteLine ("Starting Heartbeat Listener");

  this.Server = new TcpListener (IPAddress.Any, 13000);
  this.Server.Start ();

  new Thread (() => {
    // Loop forever.
    while (true) {
      if (this.Server.Pending ()) {
      // Get client and start handling thread.
      TcpClient heartbeat = this.Server.AcceptTcpClient ();

      this.HandleHeartbeat (heartbeat);
    }
    }
  }).Start ();
}
```

In order to avoid blocking code the heartbeat server runs in its own thread. This allows for the processing of node commands whilst still keeping the node information up to date. The heartbeat client implemented on the slave nodes works as defined in listing 5.7:

Listing 5.7: Heartbeat Transfer Implementation

```
1  private void SendHeartbeat (object source, ElapsedEventArgs e)
2  {
3    // Prevents any more heartbeats being sent.
4    this.HeartbeatTimer.Stop ();
5    // Creates a new TCP client on the correct port.
6    this.HeatbeatClient = new TcpClient (this.MasterNode.ToString (),
         HeartbeatPort);
7    // Use 'using' to auto dispose of client once complete.
8    using (this.HeatbeatClient) {
9      using (NetworkStream stream = this.HeatbeatClient.GetStream ()) {
10       // Generate our node meta data.
11       NodeData node = new NodeData ();
12       // Serialise the node data to a comma separated string.
13       byte[] data = Encoding.ASCII.GetBytes (node.ToSerializedPacket ());
14       // Write it to the TCP stream.
15       stream.Write (data, 0, data.Length);
16     }
17   }
18
19   // Restart the timer to allow further heartbeats.
20   this.HeartbeatTimer.Start ();
21 }
```

5.6 Implementation Issues

Below are any issues encountered within the implementation process of the system. They will each be listed with an explanation of how they affected implementation.

5.6.1 Thread Blocking Code in C#

One issue which we encountered was that much of the System library's functionality in C# uses blocking code. By this we mean that the execution of certain functions pauses the main thread disallowing any other threads to continue to process. The main example of this being in the command-line interface where functions such as **Console.ReadLine()** which is considered fairly imperative for the operation of console applications like ours.

The way to resolve this issue was to create our own functionality for handling input commands from the console. Instead of reading a line from the console, which causes the block, we simply run the **Console.ReadKey()** function in an infinite loop which only processes the completed string once it detects the return key. Using this method no other functionality is prevented by allowing user input through the console. This is implemented in listing 5.8:

Listing 5.8: Non-Blocking Console Implementation

```
1  // Loop indefinitely.
2  while (true) {
3    // Prompt user for command.
4    Console.Write ("-> ");
5    string command = string.Empty;
6    ConsoleKeyInfo key = Console.ReadKey ();
7
8    // Loop indefinitely.
9    while (true) {
10     // If the enter key has been pressed.
11     if (key.Key == ConsoleKey.Enter) {
12       // Split command into sections.
13       string[] splitCommand = command.Split (' ');
14
15       // Read command.
16       switch (splitCommand [0]) {
17         default:
18   ...
```

5.6.2 Variances in Windows and Linux

The implemented system was required to work both on Windows and Linux platforms through the use of Mono. As a result of this we get certain errors when accessing files due to the differences between Windows and Linux.

For example, on a Linux distribution the root directory of the file system is represented as '\' where windows uses 'C:/'. In our project, files are stored in the 'dfs' folder of the root directory meaning that the master node must be aware of which folder to use and create file location strings accordingly.

This has been resolved by using the **Environment.OSVersion.Platform** property built into the C# system library. This allows us to use an enumeration of operating systems in order to identify the one currently running. With a simple switch statement the master node is perfectly able to identify what operating system the node is running on. This is implemented in listing 5.9:

Listing 5.9: Platform Determination Implementation

```
switch (Environment.OSVersion.Platform) {
  case PlatformID.Unix:
    this.DfsLocation = @"/dfs/";
    break;
  default:
    this.DfsLocation = @"C:\dfs\";
    break;
}
```

5.6.3 From Scratch Genetic Algorithms vs Libraries

Genetic Algorithms can be fairly complex to implement from scratch as they require a fundamental understanding of their design and operation. Due to the time constraints placed for the project it was deemed a smarter choice to use a third-party library 'GeneticSharp' which provided the algorithmic framework. This allowed us to create the algorithm much faster and without as many errors or bugs that we would have had during development in comparison to developing the entire genetic evaluation from nothing.

However, this does not provide us with much of an understanding on how the algorithm itself functions within the library. Although this library's code is open source, the time frame of the project will not allow its inner-mechanisms to be fully understood.

5.6.4 Lack of Coding Standards

Due to time constraints it is considered that the code-base for our proposed project is fairly messy and does not conform to what many would consider to

be 'well-formed' C# code. It does not use much inheritance or interfaces like you would expect to see in an enterprise piece of software. This caused many difficulties in the later stages of development when code coherency is most critical in order to produce efficient code within the system.

In the future, it would be beneficial to re-factor the software and create more maintainable and readable code in order to aid future developments (if any) on the project.

5.6.5 Learning Curve on Ant-Colony and Genetic Algorithms

Unfortunately, due to a misunderstanding of the fundamental functionality of genetic algorithms. The ant-colony optimisation and genetic algorithms could not be implemented. This is due, in part, by the method in which the master node distributes files around the system.

In our implemented scenario, nodes are ranked in order to determine their fitness. When we attempted to implement the genetic algorithm or ant-colony optimisation we found that the problem we are attempting to determine a solution for is not considered to be an 'NP' problem. These problems are ones that can be solved in polynomial time using a non-deterministic Turing machine. By this we mean that the solution has to be testable in poly time.

The reason why genetic algorithms are used in order to solve 'NP' problems is that they cannot be solved within polynomial time in any known way. In most cases where we attempt to solve these problems without the use of a genetic algorithm, solutions are only approximate and require a metric to determine the approximation of our solution.

As our master node attempts to rank nodes instead of determining a route for files to travel, it would seem that our problem domain does not require any

kind of genetic or ant-colony algorithm in order to be solved and the use of such algorithms is deemed inappropriate for the issue. Because of this, we were not able to implement such algorithms.

However, if we were to change the network architecture so that instead of the master node communicating with a client about where the 'best-fit' nodes are, we implement a system by which the data nodes are made aware of the next node to use and they subsequently replicate the file to whichever node is next in the route instead of the client transmitting all files, we would be able to implement the two missing algorithms as they would be required to determine a route as opposed to a rank.

5.7 Testing

Our acceptance tests have been designed to cover all requirements implemented within our project. They cover the vital aspects of the system and focus on the resource management algorithms which determine node location and replication of files. Due to the nature of this software it will be impossible to find all bugs which exist within the code before our project is completed. There are still some errors which may occur afterwards. As such it will be impossible to test every single facet of the system in question, but our acceptance tests make an effort to document the test results of implemented features considered most vital.

5.7.1 Test Environment

In order to test our system, we must defined the environment in which it will be deployed. We currently test the distributed system across six Raspberry Pi microcomputers using one as a master and the rest as slaves. Each node is connected by Ethernet to a network switch which allows them to communicate. The network switch is then connected to a router in order for each node to have full network access.

5.7.1.1 Hardware Specification

In the figures below we have shown images of the environment that has been set up to facilitate testing. Specifically, we use:

- 6x Raspberry Pi 2 Model B Microcomputers
- 1x TP-LINK 8 Port Network Switch
- 1x Virgin Media Router

Figure 5.1: The Raspberry Pi Microcomputer

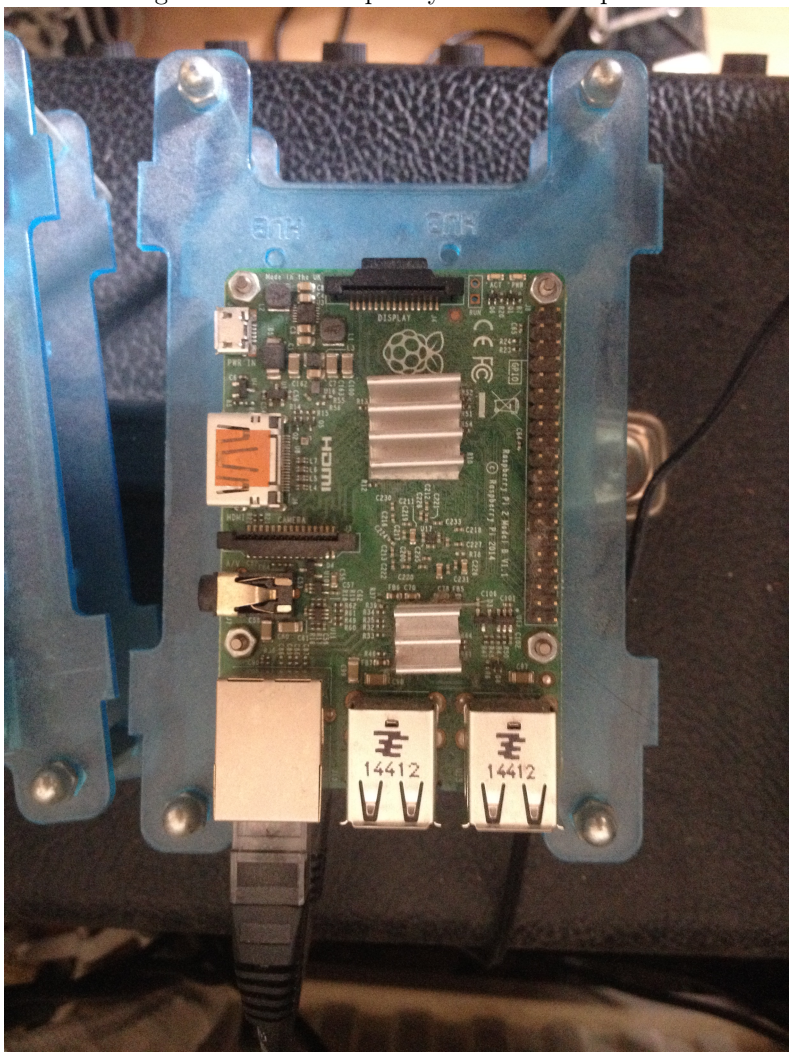


Figure 5.2: The Network Switch

Figure 5.3: The Network Router

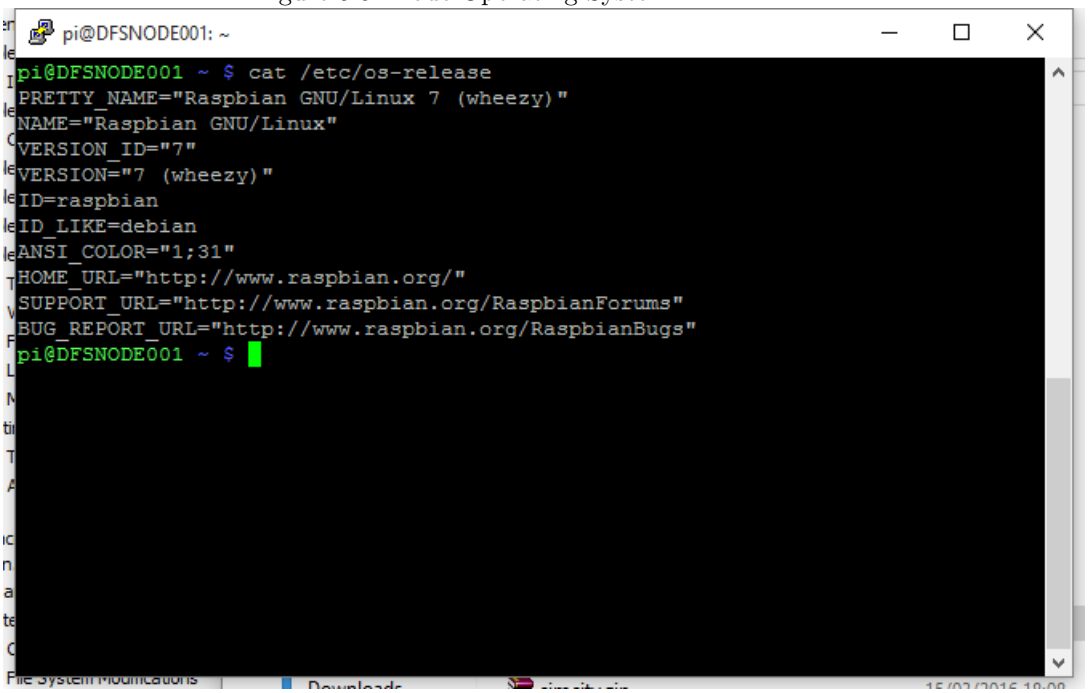
Figure 5.4: Ethernet Connections from Nodes

Figure 5.5: Node Stack Arrangement

5.7.1.2 Platform Information

Each Raspberry Pi during testing will run the Raspbian Wheezy operating system. In figure 5.5 we run the *cat /etc/os-release* command which allows us to view the operating system details for the sake of scrutiny.

Figure 5.6: Node Operating System



5.7.2 Acceptance Testing

Below are all acceptance tests and whether they passed or failed. Any failed tests will be discussed and a solution is suggested. See Requirements section for acceptance test definitions.

5.7.2.1 File Operations

Table 5.1: File Operation Acceptance Test Results

Tester	David Bond	
Test Date	14/03/2016	
Test No.	**Pass/Fail**	**Comments**
1	PASS	The file is successfully added to the file system.
2	PASS	The file is successfully removed from the file system.
3	FAIL	Due to time constraints, the ability to modify files was not implemented.

5.7.2.2 Directory Operations

Table 5.2: Directory Operation Acceptance Test Results

Tester	David Bond	
Test Date	15/03/2016	
Test No.	**Pass/Fail**	**Comments**
1	PASS	The directory is added to the cluster successfully.
2	PASS	The directory is removed from the cluster successfully.
3	FAIL	Due to time constraints, the ability to modify directories was not implemented.

5.7.2.3 Resource Management

Table 5.3: Resource Management Acceptance Test Results

Tester	David Bond	
Test Date	16/03/2016	
Test No.	**Pass/Fail**	**Comments**
1	PASS	The data node successfully connects to the master node.
2	PASS	The master node displays a list of data nodes connected.

5.7.2.4 Node Choice Algorithms

Table 5.4: Node Choice Algorithms Acceptance Test Results

Tester	David Bond	
Test Date	17/03/2016	
Test No.	**Pass/Fail**	**Comments**
1	FAIL	Due to time constraints and a misunderstanding of the problem domain, the ability to choose the genetic algorithm was not implemented.
2	FAIL	Due to time constraints and a misunderstanding of the problem domain, the ability to choose the ant-colony algorithm was not implemented.
3	PASS	The algorithm is chosen correctly and a file is added to the cluster.

Chapter 6

Results and Evaluation

6.1 System Metrics

Below are graphs and data detailing the speeds at which the distributed file system operates based on a number of different factors. The cluster is benchmarked by how quickly a file is distributed into the cluster against how large the file is. At all possible points these metrics will be compared with that of existing file systems.

6.1.1 Cluster Initialisation

These metrics will state how long the cluster takes to be fully operational, including the time taken in order to add a node to the cluster against existing number of nodes.

6.1.1.1 Starting the Cluster

The start up times were tested based on OS (both Windows and Linux). Each metric was calculated 3 times in order to produce an average.

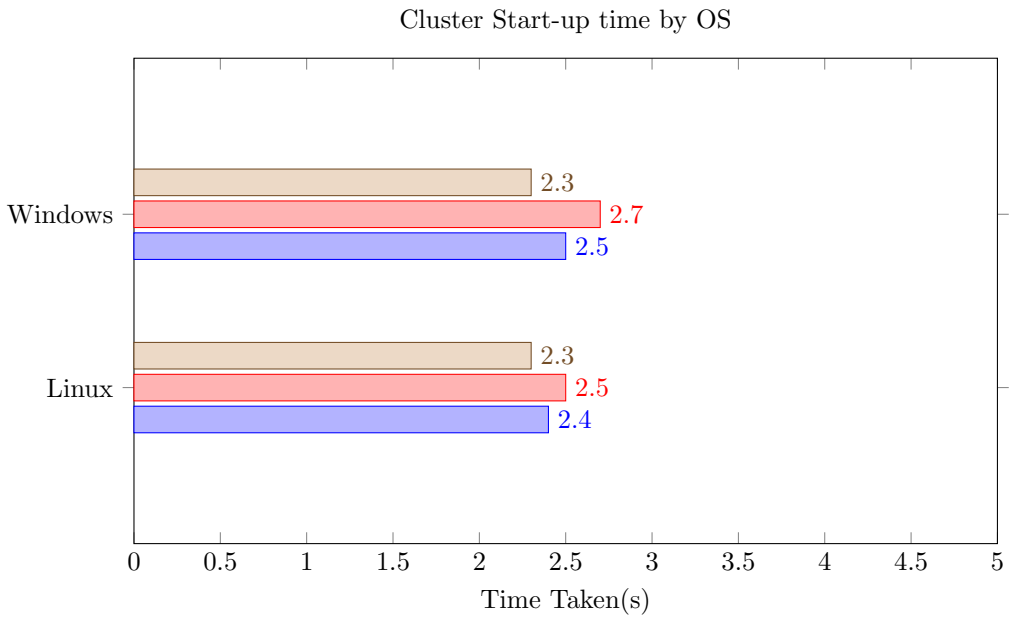


Table 6.1: Start-up time by OS

Operating System	**Run #**	**Time (s)**	**Average Time (s)**
Windows	1	2.3	2.5
	2	2.7	
	3	2.5	
Linux	1	2.4	2.4
	2	2.5	
	3	2.3	

6.1.1.2 Adding a New Node

The node addition times were tested based on operating system. Each metric was calculated 3 times in order to produce an average.

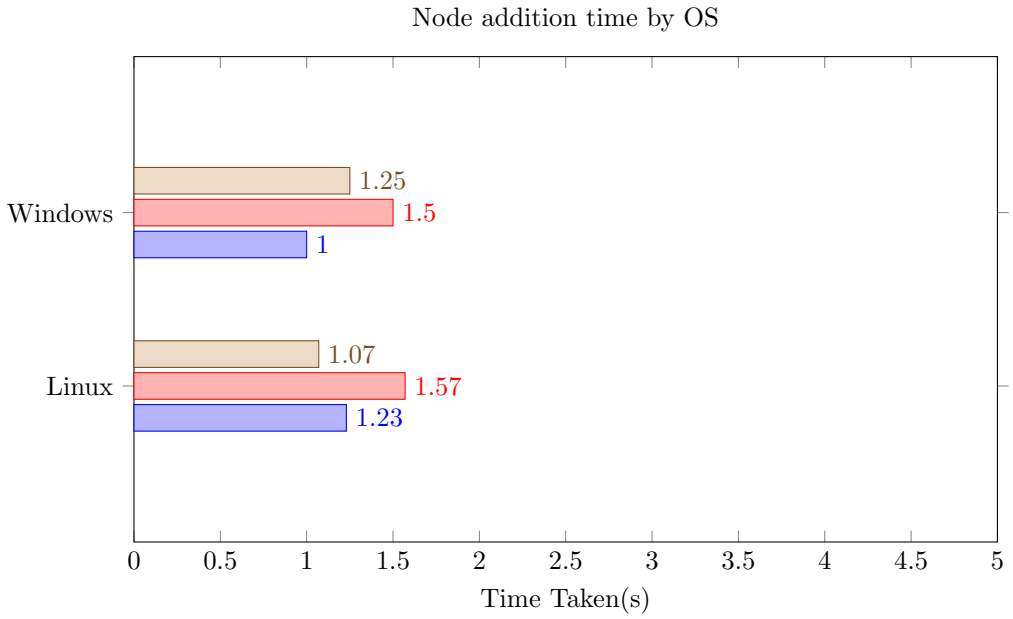


Table 6.2: Node addition time by OS

Operating System	**Run #**	**Time (s)**	**Average Time (s)**
Windows	1	1.25	1.25
	2	1.5	
	3	1	
Linux	1	1.23	1.29
	2	1.57	
	3	1.07	

6.1.2 File System Modifications

The file operation times were tested based on operating system and file size. Each metric was calculated three times using file sizes of 1, 5 and 10 megabytes. The test was also performed based on the number of nodes in the cluster. This was tested on the Windows operating system.

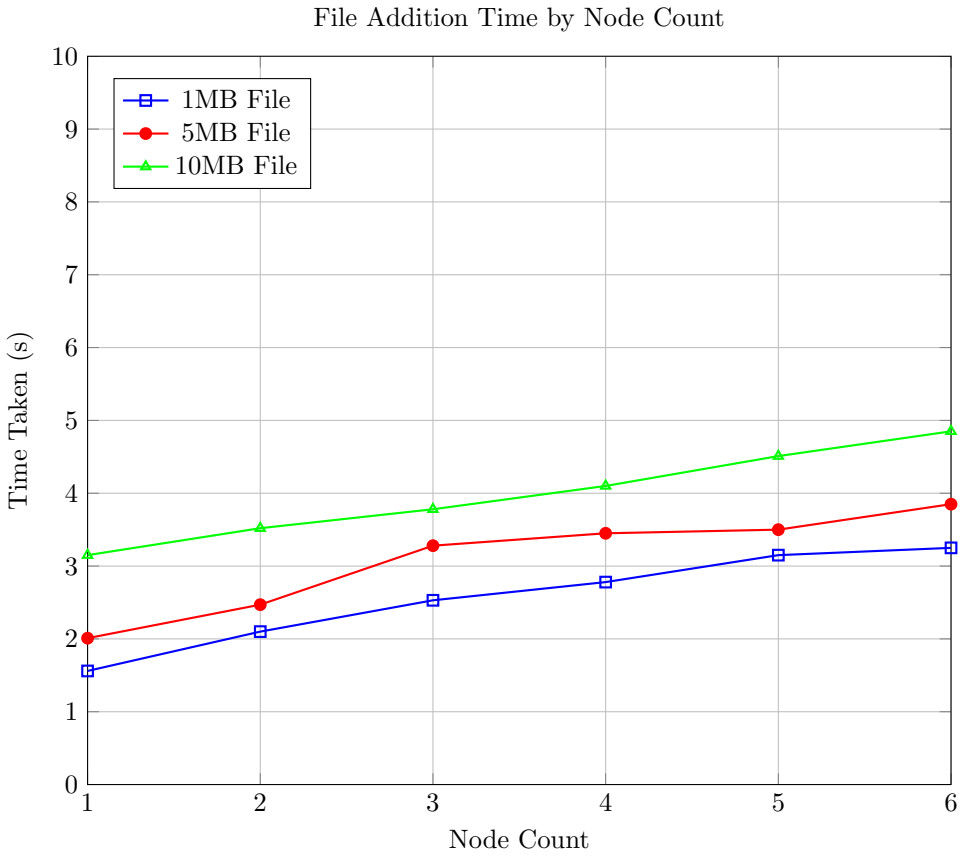


Table 6.3: File Addition Time by Node Count

File Size (MB)	Node Count	Time Taken (s)
1	1	1.56
	2	2.1
	3	2.53
	4	2.78
	5	3.15
	6	3.25
5	1	2.01
	2	2.47
	3	3.28
	4	3.45
	5	3.50
	6	3.85
10	1	3.15
	2	3.52
	3	3.78
	4	4.1
	5	4.51
	6	4.85

6.1.3 Node Selection Metrics

The node selection algorithm will be measured by which nodes it choose and how long its evaluations take. In theory, an algorithm which produces an evenly distributed selection whilst keeping its evaluation time low should be deemed a suitable choice for such a system. We will deduce this by adding 10 files to the system of equal size and measuring which nodes are chosen and how long it takes. These metrics were recorded on the Windows operating system only.

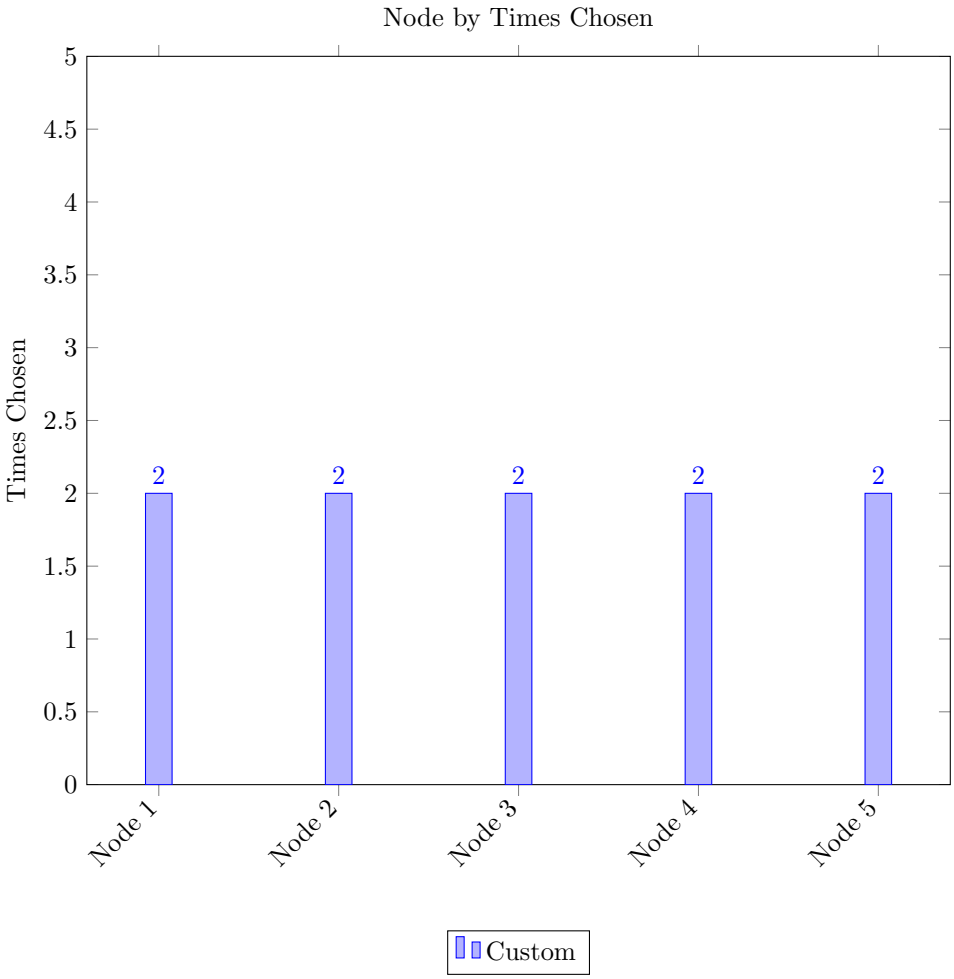


Table 6.4: Node by Times Chosen

Algorithm Type	**Data Node #**	**Times Chosen**
Custom	1	1
	2	2
	3	2
	4	2
	5	2

6.2 Evaluations

Here we will evaluate our implemented system based on its ability to transfer and distribute files compared to that of existing distributed systems.

6.2.1 Throughput & Average IO Rate

In this section we will determine the throughput a of our developed system and compare it to those of existing distributed file systems. Throughput is defined as the amount of material or items passing through a system or process. In our case, materials or items are files and is measured in megabytes per second.

Throughput is calculated using the following equation:

$$Throughput(N) = \frac{\sum_{i=0}^{N} filesize_i}{\sum_{i=0}^{N} time_i} \tag{6.1}$$

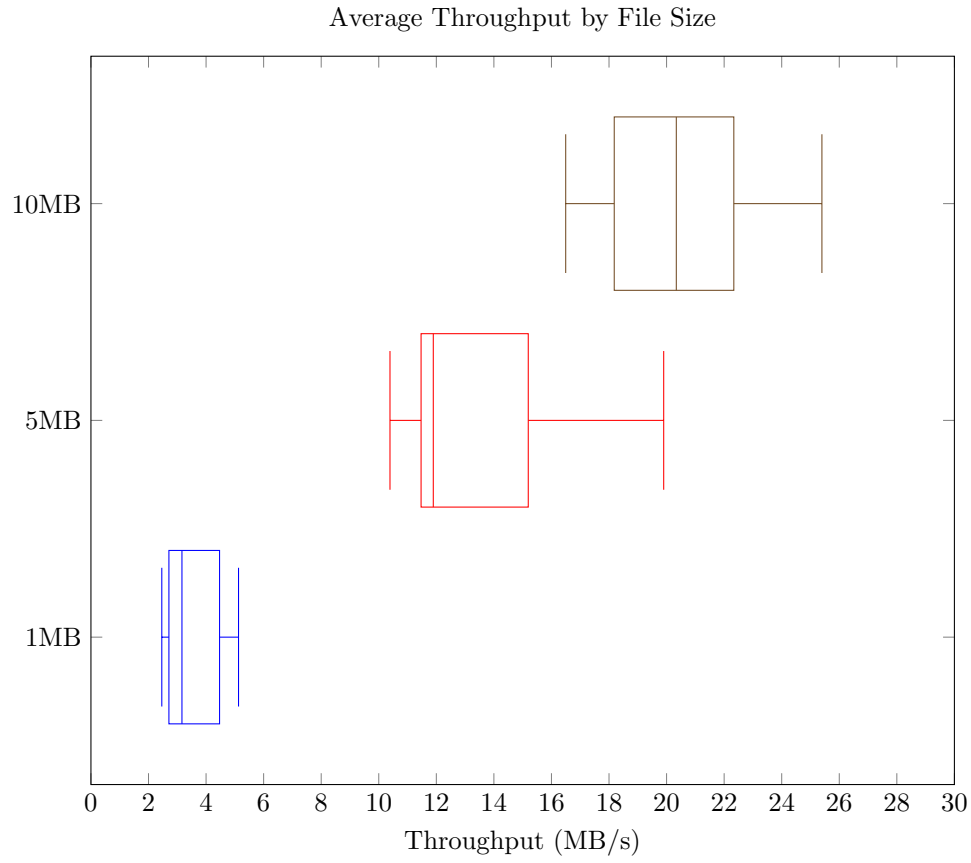
In implementation the file system Hadoop has shown a high throughput making it one of the standard choices for distributed systems today. In 2009 performance tests were taken producing average results of 33 to 34MB/s on most operations in the system[24].

6.2.1.1 Throughput Results

Based on these equations and the data collected in section 6.1.2, we can estimate the system's throughput based on how many nodes are in the system using our

equation above.

On average, the results for a 1MB file transfer demonstrates a throughput of 3.33MB/s, with the upper limit being 5.128MB/s and the lower limit being 2.462MB/s. When we test the throughput using a 5MB file, we see results on average of 13.617MB/s, with an upper limit of 19.9MB/s and a lower limit of 10.39MB/s. Finally, when we use the data from our 10MB file transfer we see an average throughput of 20.5055MB/s with an upper limit of 25.397MB/s and a lower limit of 16.495MB/s. This data is represented in the following box plot:



This data shows a trend which is commonly found in file systems today. As the number of nodes increases in the system, it will take longer for a file to be

Table 6.5: Average Throughput by File Size

File Size	Median	Upper Quartile	Lower Quartile	Upper Whisker	Lower Whisker
1 MB	3.162	4.469	2.709	5.128	2.462
5 MB	11.8945	15.19425	11.47025	19.9	10.39
10 MB	20.338	22.33625	18.1815	25.397	16.495

fully distributed using non-random placement. It seems as though the implementation of a decision algorithm to perform file placement hinders throughput, where a system which prioritises random file distribution (such as Hadoop) is able to maintain a high throughput as the system does not take much care as to which particular nodes in the cluster files are sent to.

However, the metrics we have collected here are for relatively small file sizes. Distributed systems are typically implemented in order to handle vast amounts of information in a short period of time. In future, it would be best to test the system using larger sizes, preferably expanding into the gigabytes and tens of gigabytes.

At this point in time it does not seem as though our testing methods have hit the limit of what throughput values the system may produce. However, from the data collected we can see that the number of nodes in the system has a negative relationship with that of the throughput, increasing the time it takes for the system to make a decision on where to place a file.

6.2.1.2 Comparison to HDFS

In order to evaluate our system we must compare it to competing distributed file systems. In this section we will discuss how our system compares to that of HDFS, the main metric we will be using for comparison is throughput (see section 6.2.1).

The graph below shows how our system's average throughput compared to

that of HDFS. The data was collected from the 2009 evaluation of Hadoop performed by Tien Duc Dinh[24]. Their report shows throughput results for three file addition operations.

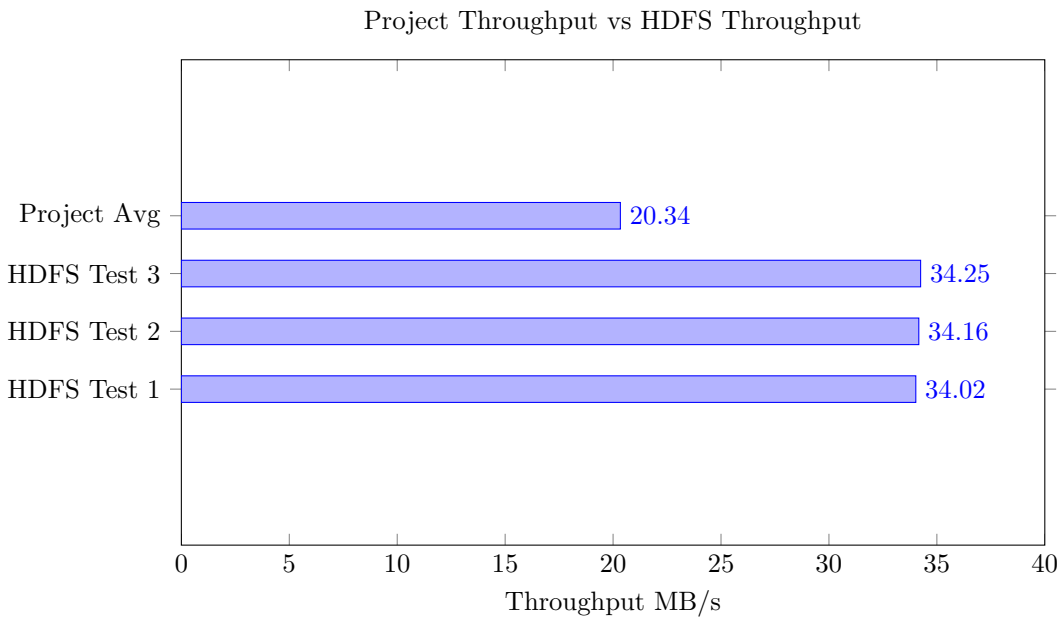


Table 6.6: Project Throughput vs HDFS Throughput

Test Platform	Throughput (MB/s)
HDFS Test 1	34.024
HDFS Test 2	34.163
HDFS Test 3	34.247
Project Avg	20.338

As previously discussed, Hadoop prioritises random file placement in order to keep throughput high. Based on the collected data above it seems that our implementation of a decision algorithm may have had an impact on throughput in comparison to HDFS. In fact, our system displayed a throughput approximately 14MB/s slower than that of the three HDFS tests.

This implies that our algorithm to rank nodes may require some optimisation in the future in order to produce better throughput results. In subsequent implementations of the system, we may be able to test the throughput results of the genetic and ant-colony algorithms.

6.2.2 File Distribution

Based on the results shown in section 6.1.3, it is shown that our implemented algorithm shows great promise when distributing files. Although the amount of files given would not constitute a system such as Hadoop (which has been tried and tested with huge files), we can see that our master node makes good decisions about where files should be placed. Out of the 10 files added, each of the five nodes was given two files, creating a smooth distribution across all nodes.

This also demonstrates that the node heartbeat system works well within the master and data nodes to ensure that the meta data used to determine best fit nodes is up-to-date, allowing best-fit calculations to be very accurate.

In future, it would be best to test the system using much higher file sizes and many more files. This will allow us to create a much more detailed idea of how well the system will distribute and give accurate figures on demonstrated throughput.

Chapter 7

Future Work & Conclusions

7.1 Learning Outcomes

Overall, the project was a moderate success. The important features of the distributed file system were all included and the individual algorithms were also successfully implemented. In retrospect, having a modern, formal software development process such as Agile would have potentially increased the project's productivity and could have led to the remaining features being implemented.

Ultimately, much knowledge has been gained into the technologies behind a distributed file system and completion of the project has shown that moving towards autonomous based systems can lead to improvements in operational ability.

In completing the project there were occasions when some requirements remained incomplete due to the more complex nature of the system being proposed, which, with hindsight, might have been more appropriately scheduled. From a technical point of view I have discovered and understood a number of networking features in C# that I was previously unaware and all of the techniques used and acquired during the project and are fully transferable skills that I can take into other fields.

I've also gained a much deeper understanding in how the software architecture for such a system should be produced and all the different features that are required to have both a usable and operational distributed file system. I've also gained much insight into the mechanisms which make other distributed systems optimal and functional during my research stages.

7.2 Looking Forward

In this section we will discuss future improvements which could be made to the produced system in order to turn it into a viable product for real-world

implementation. Many of these features would have been possible through the use of a stricter development process but were unfortunately left out.

7.2.1 File Chunking

The created file system in its current state only really demonstrates the ability for a master node to determine where file should be sent based on multiple factors. Although this optimisation is necessary there are still operations which can be performed at the file level to ensure greater resource management across the entire system.

One example of these optimisations could be the chunking of files performed on a new node. 'Chunking' is the act of taking the binary data of a file and separating it into several smaller files through the use of buffers. The main benefit this brings to the system is that in the case of data loss, only chunks of the files are lost. If suitable replication has occurred, less network usage is required and a file can be restored asynchronously across several nodes, leading to a quicker time to restore the file fully.

GFS (described in the Background section) utilises a similar system in order to preserve its files fully. Specifically they are separated into 64MB chunks[6] which are rarely overwritten or shrunk, successfully increasing the redundancy of their data.

7.2.2 Encryption & Security

Currently, the implemented system sends all network traffic in an unencrypted, serialised form which means that messages between nodes or even entire files could be read through the use of a packet sniffer. In modern business enterprises (which may use a file system such as this) data needs to be kept secure in order for companies to protect their intellectual property. In any future iterations of the file system created, network encryption is to be considered necessary.

One method of implementing this could be through the use of .NET's **'System.Security.Cryptography'** namespace which allows access to several classes and methods which can encrypt data in different ways depending on the hardware architecture. One such class is the **'CryptoStream'** class which links data streams to cryptographic transformations, providing an interface which allows data to be encrypted/decrypted as it is streamed between machines.

7.2.3 Change of Programming Language

Typically, systems which require complex networking and handling of file data use more verbose languages which give a lower level of memory access such as C++. In future, it may be a better solution to rewrite the application using a lower level language.

However, C# provides a much simpler interface with a language that is closer to written English than C++. C# also provides many classes and functions by default in the *System* library which make developing this kind of application much easier. For example, in C++ a typical TCP server and client configuration could span several functions across many classes. C# on the other hand provides a *TcpClient* class which can run asynchronously by default.

Asynchronism can be a difficult task to implement in C++ as it requires hefty use of threading. C# stepped around this issue by using *async* and *await* identifiers to tell the compiler how to handle threads. Overall, if the project had a larger time frame it would be better to create the application in a lower level language which can be tweaked and optimised fully.

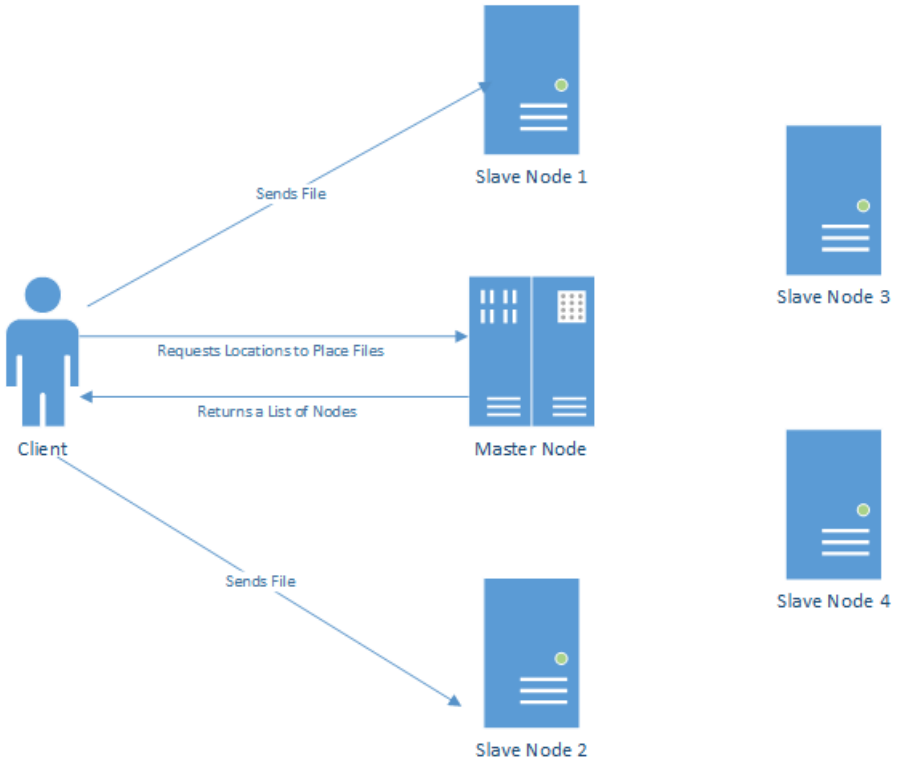
7.2.4 Modification of Architecture to Solicit a Genetic Algorithm

In section 5.6.5 we discuss the implementation issue of having the wrong problem domain to suit a genetic or ant-colony algorithm. This is primarily to the

network infrastructure chosen at the start of our project. In this section, we will discuss what changes should be made to the network architecture to allow incorporation of such an algorithm in future revisions of the software.

To begin, figure 7.1 below demonstrates how data is passed around the system in our current implementation. In this figure we can see that the client requests locations of nodes on which to store files. Once the client has obtained this list it will send the files one at a time to each respective data node. In this situation, the master node needs only to produce a best fit node for each time the client wants to place the file. Unfortunately, this system does not create an 'NP' problem for which a genetic algorithm would be useful.

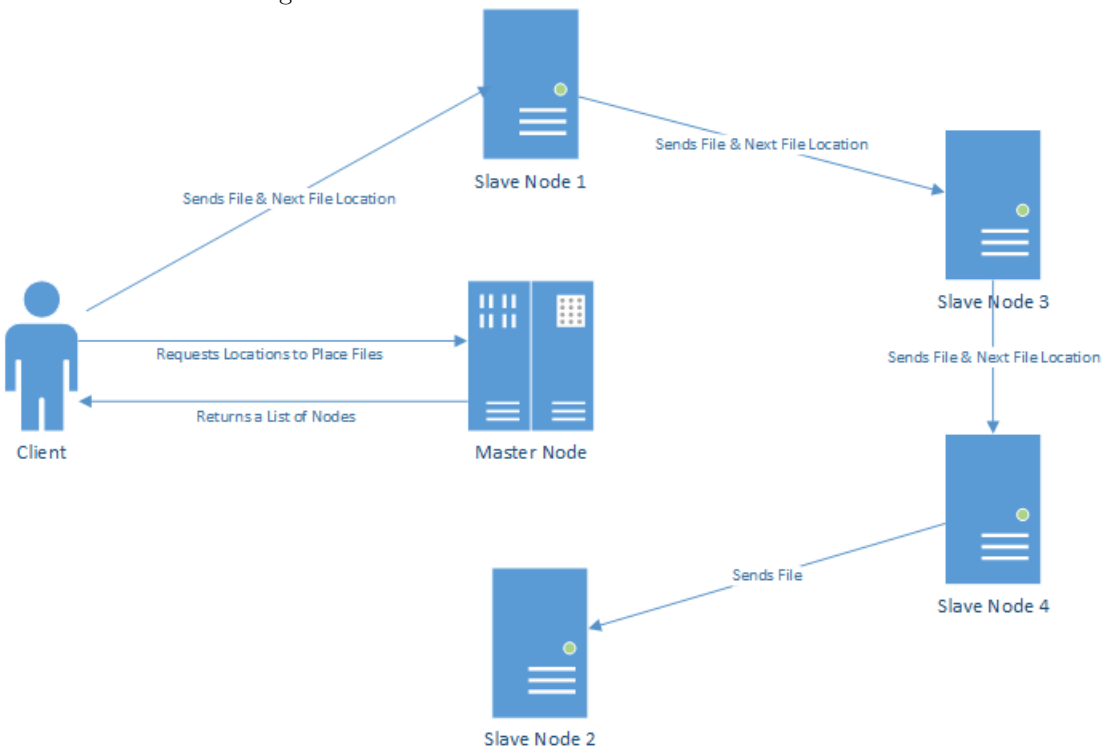
Figure 7.1: Current Network Flow



However, if we were to modify the system to reflect figure 7.2 below, we would be able to implement a genetic algorithm as the problem moves away

from ranking and towards routing. If the nodes have to determine a route to take then we can apply a genetic or ant-colony algorithm to solve the issue.

Figure 7.2: Potential Network Flow



7.3 Project Summary

Overall, the project was deemed a moderate success. Although not all algorithms discussed were able to be implemented, we were still able to create a distributed file systems which makes decisions on where files should be placed based on meta data collected from nodes. Although our implemented solution appears to distribute files well, it will require future work in order to determine if a different kind of algorithm would work more efficiently.

7.3.1 University Modules which Benefited the Project

Over the course of my studies I have undertaken several modules which have helped me in the development of this project. Firstly, the C & C++ programming modules contained within the first and second years allowed me to build the foundations of my programming knowledge and taught me the fundamentals of writing software.

Secondly, the 'Computer and Network Systems' & its counterpart 'Computer Networks and Operating Systems' in the first and second years of the degree helped teach me the fundamentals of networking. Including writing software for communicating over TCP which was used widely in the project. Without this knowledge my project may not have been possible.

Thirdly, the choice to use the programming language C# came from my extensive use of the language during my placement year. Without the skills I obtained throughout that year I don't believe it would have been possible for me to develop such a system.

Overall, there are some degree modules which I undertook that have directly influenced the development of a distributed file system.

Chapter 8

Glossary

8.1 Abbreviations and Initialisms

Below I have listed abbreviations and initialisms used throughout the project, and defined them in alphabetical order:

- CLI - Command line interface
- EAI - Enterprise application integration
- GB - Gigabyte
- GUI - Graphical User Interface
- HDFS - Hadoop File System
- HTML - Hyper Text Markup Language
- Hz - Hertz
- MHz - Megahertz
- MoScoW - Must, should, could, and would have
- NAS - Network Attached Storage
- NFS - Network File System
- OSI - Open Systems Interconnection
- POI - Point of Interaction
- RAID - Redundant Array of Independent Disks
- RAM - Random Access Memory
- SSH - Secure Shell
- TB - Terabyte
- USB - Universal Serial Bus
- VC - Video Codec
- W3C - World Wide Web Consortium
- XML - eXtensible Markup Language

Chapter 9

References

Bibliography

[1] Amina Saify, Garmina Koxhar, Jenwei Hsieh & Onur Celebidglu: *Enhancing High-Performance Computing Clusters with Parallel File Systems, 2005.*

http://www.dell.com/downloads/global/power/ps2q05-20040179-Saify-OE.pdf

[2] The Apache Software Foundation: *HDFS Architecture Guide, 2015.*

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[3] Benjamin Atkin & Kenneth P. Birman: *MFS: an Adaptive Distributed File System for Mobile Hosts, 2015.*

https://www.cs.cornell.edu/batkin/docs/mfs.pdf

[4] Mathew J. Schwartz: *Java Under Attack Again, Disable Now, 2013.*

http://www.darkreading.com/attacks-and-breaches/java-under-attack-again-disable-now/d/d-id/1108136?

[5] Josh Lerner & Jean Tirole: *The Simple Economics of Open Source, 2000.*

http://www.nber.org/papers/w7600.pdf

[6] Sanjay Ghemawat, Howard Gobioff & Shun-Tak Leung: *The Google File System, 2003.*

http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

[7] Frank Schmuck & Roger Haskin: *GPFS: A Shared-Disk File System for Large Computing Clusters, 2002.*

https://www.usenix.org/legacy/events/fast02/full_papers/schmuck/schmuck.pdf

[8] Veera Deenadhayalan: *General Parallel File System (GPFS) Native RAID, 2011.*

https://www.usenix.org/legacy/events/lisa11/tech/slides/deenadhayalan.pdf

[9] IBM: *GPFS Official Documentation - Disk Hospital, 2015.*

https://www-01.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs200.doc/bl1adv_introdiskhospital.htm

[10] Dino Quintero, Matteo Barzaghi, Randy Brewster, Wan Hee Kim, Steve Normann, Paulo Queiroz, Robert Simon & Andrei Vlad: *Implementing the IBM General Parallel File System (GPFS) in a Cross Platform Environment, 2011.*

https://books.google.co.uk/books?id=6hnCAgAAQBAJ

[11] Apache Software Foundation: *Hadoop Cluster Setup, 2015.*

https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html

[12] Todd Papaioannou: *Solving Big Data App Developers' Biggest Pains, 2013.*

https://www.youtube.com/watch?v=dWagSVBkaJk&w=560&h=315

[13] Apache Software Foundation: *MapReduce Parameter List, 2015.*

https://hadoop.apache.org/docs/r2.7.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml

[14] Jeff Sposetti & Apache Software Foundation: *Install Ambari 2.2.0 from Public Repositories, 2015.*

https://cwiki.apache.org/confluence/display/AMBARI/Install+Ambari+2.2.0+from+Public+Repositories

[15] Marco Dorigo & Thomas Stützle: *Ant Colony Optimization: Overview and Recent Advances, 2009.*

https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/DorigoStuetzle09%20-%20ACO%20overview%20and%20recent.pdf

[16] Marco Dorigo & Alberto Colorni: *The Ant System: Optimization by a colony of cooperating agents, 1996.*

https://www.researchgate.net/profile/Vittorio_Maniezzo/publication/5589170_Ant_system_optimization_by_a_colony_of_cooperating_agents/links/00b7d516470a53e6d0000000.pdf

[17] Melanie Mitchell: *An Introduction to Genetic Algorithms, 1996.*

http://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf

[18] Glen Corneau: *IBM Spectrum Scale Lab Exercises: Using Replication, 2012.*

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Using%20Replication

[19] Prof. Dr. Riko Šafaric: *Intelligent Control Techniques in Mechatronics, 2012.*

http://www.ro.feri.uni-mb.si/predmeti/int_reg/Predavanja/Eng/3.Genetic%20algorithm/index.html

[20] IBM: *General Parallel File System Administration and Programming Reference, 2012.*

http://publibfp.dhe.ibm.com/epubs/pdf/a2322215.pdf

[21] M. Kornacker & J. Erickson: *Cloudera Impala - Real Time Queries in Apache Hadoop, For Real, 2012.*

http://blog.cloudera.com/blog/2012/10/cloudera\-impala-real-time-queries-in-apache-hadoop-for-real/

[22] Otman Abdoun & Jaafar Abouchabaka: *A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem, 2011.*

http://arxiv.org/ftp/arxiv/papers/1203/1203.3097.pdf

[23] Otman Abdoun, Jaafar Abouchabaka & Chakir Tajani: *Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem, 2011.*

http://arxiv.org/ftp/arxiv/papers/1203/1203.3099.pdf

[24] Tien Duc Dinh: *Hadoop Performance Evaluation, 2009.*

https://wr.informatik.uni-hamburg.de/_media/research/labs/2009/2009-12-tien_duc_dinh-evaluierung_von_hadoop-report.pdf