Bachelor Degree Project

TRANSPARENCY ANALYSIS OF DISTRIBUTED FILE SYSTEMS

With a focus on InterPlanetary File System

Bachelor Degree Project in Information Technology
Basic level 30 credits
Spring term 2018

Oscar Wennergren, Mattias Vidhall, Jimmy Sörensen

Supervisor: Jonas Mellin

Examiner: Joe Steinhauer

Abstract

IPFS claims to be the replacement of HTTP and aims to be used globally. However, our study shows that in terms of scalability, performance and security, IPFS is inadequate. This is a result from our experimental and qualitative study of transparency of IPFS version 0.4.13. Moreover, since IPFS is a distributed file system, it should fulfill all aspects of transparency, but according to our study, this is not the case.

From our small-scale analysis, we speculate that nested files appear to be the main cause of the performance issues and replication amplifies these problems even further.

Keywords: Distributed file systems, replication, performance, scalability, security, transparency

Table of Contents

1 Introduction 1

2 Background 2

2.1 Recommendations for IPFS configuration and usage 2

2.2 Distributed File Systems 2

2.2.1 Peer-to-peer 2

2.2.2 Client/Server 3

2.3 File systems under study 3

2.3.1 InterPlanetary File System: A brief overview 3

2.3.2 Network File System 3

2.3.3 ext4 3

2.4 Aspects of Transparency in Distributed File Systems 3

2.5 Reasons for studying IPFS 4

2.6 System attributes 7

3 Problem definition 8

3.1 Aim 8

3.2 Motivation 8

3.3 Research Question 8

3.4 Objectives 8

3.5 Hypothesis 9

3.6 Areas of Responsibility 9

4 Methodology 10

4.1 Shared experimental settings for Scalability and Performance 10

4.2 Scalability 11

4.2.1 Chosen Strategy 11

4.2.2 Method Implementation 11

4.2.3 Dependent and independent variables 12

4.3 Performance 13

4.3.1 Chosen Strategy 13

4.3.2 Method Implementation 13

4.3.3 Dependent variables 16

4.4 Security 17

4.4.1 Chosen Strategy 17

4.4.2 Method Implementation 17

4.5 Qualitative analysis of subjective aspects of transparency 19

4.5.1 Access Transparency 19

4.5.2 Location Transparency 19

4.5.3 Failure Transparency 19

4.5.4 Migration Transparency 19

4.6 Handling of validity threats 20

4.6.1 Conclusion Validity 20

4.6.2 External and Internal Validity 21

4.6.3 Construct Validity 21

4.7 Alternative methodological strategies 22

5 Related work 23

5.1 Scalability 23

5.2 Performance 23

5.3 Replication 23
5.4 Security 23

6 Evaluation 25
6.1 Scalability 25
6.1.1 Results 25
6.1.2 Analysis 25
6.1.3 Conclusion 27
6.2 Performance 28
6.2.1 Results 28
6.2.2 Analysis 29
6.2.3 Conclusion 31
6.3 Security 32
6.3.1 Results 32
6.3.2 Analysis 35
6.3.3 Conclusion 36
6.4 Qualitative analysis of subjective aspects of transparency 37
6.4.1 Analysis 37
6.4.2 Conclusion 38

7 Discussion 39
7.1 Summary 39
7.2 Ethical aspects in experimentation 39
7.3 Ethical aspects in Security 39
7.4 Experimental simulation results 40
7.5 Recommendations for IPFS 40
7.6 Small-scale analysis of IPFS 41
7.7 Security 43
7.8 Subjective aspects of transparency 44
7.9 An IPFS DDOS Attack 44

8 Future Work 46
8.1 Future work concerning both Scalability and Performance 46
8.2 Scalability 46
8.3 Performance 46
8.4 Security 46

9 References 47

1 Introduction

Distributed file systems are used to allow multiple users to access and store files on a shared file system over networks. The aim of a distributed file system is that the user should not be able to tell that the file system is distributed and not a local file system on the user's computer. This means that the file systems should not be unreasonbly slow or be more difficult to access. This is called transparency and there are multiple aspects of transparency (section 2.4) and the common goal of the aspects is that they aim to make the distributed file system "invisible" to the user.

Transparency cannot be measured objectively, as it is a perceived feeling that may differ between users. Instead, the potential transparency can be derived from comparing results. For this reason, a comparision between InterPlanetary File System (IPFS) and Network File System (NFS) was performed, with ext4 used as a baseline.

IPFS is a distributed file system that is on the rise. This file system makes use of peer-to-peer technology which allows each computer using the file system to share the workload. Due to the fact that IPFS initially started in 2014 and is still in an active development state, not a lot of research has been performed on the file system and currently there is no previous research on how transparent the file system is. According to Coulouris, Dollimore, Kindberg and Blair (2011, p.521) a distributed file system should feel comparable to local file systems in terms of performance and reliablity.

Therefore, the aim of this study is to investigate how the transparency aspects are perceived in IPFS in terms of the system attributes; scalability, performance and security. To achieve this, experiments were performed on scalability and performance and a qualitative analysis on security. Experimention was chosen as a scientific method to allow for better control of variables that could affect the outcome of the study. Qualitative analysis is also performed to analyze the aspects of transparency that are subjective.

The performance experiment draws inspiration from methods derived in previous research on distributed file systems and by performing predefined file operations on IPFS, ext4 and NFS. The performance is measured in completion time, processor and bandwidth usage of the file operations. The transparency in the file systems are then analyzed by comparing the performance results from the different file systems.

The scalability experiment was inspired from methods where research on distributed file systems was performed with varying cluster size and replication factors. These variables were not used together in previous research, thus inspired the use of both these variables. Results are then analyzed by comparing the effects of varying cluster sized and replication factors and then transparency can be determined from these results.

The security analysis draws inspiration from Coulouris *et al.* (2011), where they bring up the threats that a distributed file system should be able to handle. By performing a qualitative analysis on both file systems, NFS and IPFS, with these threats in mind and evaluating how well they handle the threats, it should be possible to make an evaluation as to which file system is the right choice for what it is needed for.

The results from this study indicate that IPFS does not fulfill the requirements for full transparency and is also inadequate in the areas we researched. For most common scenarios, we recommend NFS over IPFS.

2 Background

This chapter describes the different techniques and file systems relevant for this study, such as distributed file systems and IPFS.

2.1 Recommendations for IPFS configuration and usage

The aim of the IPFS is to link together every computing device globally, to create a permanent web with the same set of files (Benet, 2014b). However, there are no recommendations for when IPFS would be more suitable compared to other distributed file systems. Thus, one of our goals is to provide enough information so that whoever is interested in using IPFS can determine if it fits their purpose.

Currently we are seeing a rise in popularity for using IPFS for projects, most recently the biggest public use of IPFS was its use in sidestepping the censorship in Catalonia during the voting of independence (Kilburn, 2017). IPFS was used since the distributed content could be accessed via any IPFS instance with a public gateway if the nodes with the specified content was not turned off. This allowed the population of Catalonia to access the content even though the Spanish government had tried to censor this information.

2.2 Distributed File Systems

According to Coulouris *et al.* (2011, p. 521), "A distributed file system enables programs to store and access remote files exactly as they do local ones, allowing users to access files from any computer on a network. The performance and reliability experienced for access to files stored at a server should be comparable to that for files stored on local disks."

The latter is what is known as transparency. There are a couple of aspects within transparency such as performance transparency and location transparency.

There are different distributed file systems (DFS) that are used for various scenarios. Two of these are IPFS (a Peer-To-Peer based DFS) and NFS (a Client/Server based DFS), which are the two distributed file systems that are analyzed in the report.

2.2.1 Peer-to-peer

A peer-to-peer (P2P) network consists of multiple computers called nodes or peers that are connected. Peers simultaneously serve in the roles of both clients and servers. This works by each node holding small parts of the resource and when a node requests a resource it asks around the peer-to-peer network until it finds the nodes that has the information.

By spreading out the resources over several nodes in a network, it also mitigates the risk of losing data in the case of an accident, as the data is still available on other nodes in the network if one goes down.

The benefits of peer-to-peer are excellent scalability as more nodes are added to the cluster by balancing the load. The drawbacks of peer-to-peer is mainly security, it is very expensive to add access rights to the network so that changes to files can be controlled, compared to a client/server solution where the server handles all the rights.

2.2.2 Client/Server

A Client/Server network is constructed by clients sending requests to a server to access files stored on the server and depending on the request and the client's rights, the server will either respond or deny the client's request.

One drawback with a client/server architecture is that when adding more clients to the system, it will eventually be overloaded with requests, which means that more servers must be added to alleviate the load on the current servers.

This also means that the location of the data is all in one place. This increases the risk of losing the data should something happen to it, which requires setting up backup procedures for the data. At the same time, this also means that the data can be controlled, and will not easily end up in the wrong hands.

2.3 File systems under study

For the purpose of the study and to be able to objectively compare the results, we used three different file systems in our report; IPFS and NFS, two distributed file systems, and ext4, a local file system. NFS was chosen as it is a commonly used distributed file system (Coulouris et al., 2011, p. 521), while ext4 was chosen as it is the default file system in CentOS 6, which is based on Red Hat Enterprise Linux 6 (CentOS Project, 2017; Red Hat, 2017).

2.3.1 InterPlanetary File System: A brief overview

IPFS is a distributed file system making use of the peer-to-peer architecture. The goal of IPFS is to connect all computing devices to the same system of files (Benet, 2014b).

IPFS is based on Git and uses BitSwap technology for transferring files over the network. Content added to IPFS receives a unique hash corresponding to the contents of the resource (i.e. files in a folder or contents of a file). Due to this unique hash, it does not matter if the name differs, the specified content is delivered if it finds the corresponding hash.

2.3.2 Network File System

Network File System (NFS) is a distributed file system based on the client/server architecture. NFS has been widely used as a file system in both industry and academic environments (Coulouris et al., 2011, p. 521). However, this also means that if one wants to use NFS for a Wide Area Network (WAN) application, such as over the web, it loses performance and security because of the scale of clients and the number of servers that is needed.

2.3.3 ext4

Fourth extended file system (ext4) is a journaling, native, and local file system for Linux that is the successor of the third extended file system (ext3). With support for larger file sizes, more directories, more features and better accessibility (Kernel.org, 2008).

2.4 Aspects of Transparency in Distributed File Systems

Achieving high transparency in a distributed file system is desirable and these are the following aspects of transparency that have been pointed out by Coulouris *et al.* (2011, p. 23) as:

"**Access transparency**

enables local and remote resources to be accessed using identical operations.

Location transparency

enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

Concurrency transparency

enables several processes to operate concurrently using shared resources without interference between them.

Replication transparency

enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Failure transparency

enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

Migration transparency

allows the movement of resources and clients within a system without affecting the operation of users or programs.

Performance transparency

allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency

allows the system and applications to expand in scale without change to the system structure or the application algorithms."

All these aspects, except for concurrency, are analyzed in subsequent chapters of this study. Concurrency is not included, as it is a very complex subject and performing an analysis on this aspect would require verifying that multiple nodes can access the same resources at the same time, while the system still maintains the correct information. When an application or task requires concurrency, a database is better suited than a file system (Coulouris et al., 2011, p. 675).

To perform a concurrency analysis, tools for verifying synchronization would be required to evaluate that the specified file systems are working as intended. This requires a lot of time and effort, and thus not possible in the given time frame for this bachelor thesis.

2.5 Reasons for studying IPFS

IPFS uses peer-to-peer architecture and because of this, it has no single point of failure (Benet, 2014b, p. 2). IPFS combines multiple ideas from previous peer-to-peer systems such as Distributed Hash Tables (DHT), BitTorrent and Git (Benet, 2014b, p. 3). This combination in addition to the aim to replace current file distributions techniques with more modern techniques makes it an interesting file system to study. The purpose is to see how well it compares to other distributed file systems that are commonly used.(Benet, 2014b, p. 1).

Identity
Each node in IPFS has an identity that is unique to each node. When multiple nodes connect to the network they use public key cryptography to identify each node. This is to make sure that each node is authenticated and not pretending to be a different node (Benet, 2014b).

Routing
As IPFS is a peer-to-peer system the nodes all have the same privilege. Furthermore, to find all the relevant nodes in the system it uses a routing system. This system is used to determine what objects all known nodes can deliver. In IPFS this is achieved using a DHT that is based on the earlier work mentioned in Benet (2014, p. 4).

File transfer
For distributing files and data in IPFS, a protocol is used that is like that of BitTorrent. This protocol is called BitSwap and it works by peers having a "wantList" where they seek to acquire a set of blocks, and peers that have a "haveList" where it states what blocks they have to offer in exchange (Benet, 2014b).

Blocks are binary structures of data that contain a certain amount of the files information. By splitting the content up in several blocks, it is possible to fetch the entire file from several nodes at once, by allowing each node to supply the recipient with different blocks of the file (Benet, 2014b).

Merkle DAG
The DHT and BitSwap allows IPFS to grow to a massive peer-to-peer system for storing and distributing blocks. Furthermore, IPFS creates a Merkle Directed Acyclic Graph (Merkle DAG) which is a tree that is used to, as described in Benet 2014, p.6 (ConsenSys, 2016):

- Verify that each object in IPFS has its own unique hash.
- Make sure that everything that is transferred over IPFS is verified with its checksum. If data is tampered with or corrupted, IPFS will detect it.
- Check if an object has the exact same content as another object in IPFS. In that case, it is considered equal and only stored once.

Distributed Hash Tables
Distributed Hash Tables is a widely used concept used for coordinating and maintaining metadata in peer-to-peer systems. IPFS uses a Distributed Sloppy Hash Table (DSHT) as described in (Benet, 2014, p. 2).

The DSHT Implementation used in IPFS is one that protects against malicious attacks in two ways; one being securing NodeID generation, which prevents Sybil attacks. It also has a 0.85 success rate for ensuring only honest nodes can connect to each other, when there is a large fraction of adversaries in the network (Benet, 2014b, p. 4).

A Sybil attack is an attack on a reputation system where the attackers subvert it by forging new identities in a peer-to-peer system. This allows the attacker to create a lot of fake identities and gain a lot of influence in the network (Trifa & Khemakhem, 2014).

A distributed hash table is a class of decentralized distributed systems that provide easy lookup of data similar to a regular hash table (key, value). The maintaining of the table is distributed among all the nodes in such a way that if there is a change in participants, the disruption is minimal.

A DSHT allows nodes on the network to locate nearby copies of the file, regardless of how popular it is, this prevents "hotspots" in the indexing infrastructure (Freedman & Mazi, 2003).

Local storage and pinning

IPFS makes use of local storage to store and to retrieve the data that is sent on the distributed file system. In short, almost every block that is available to nodes are stored in some node's local storage.

When a node requests a file, they retrieve it from a different node and store it in their local storage as well. This allows for multiple nodes to have the possibility to deliver the specified content, with a short lookup period. However, because it is local storage it is not stored forever. If a node deems an object to be of importance, a user can pin an object to ensure the survival of the object. If an object is pinned it will stay in the local storage until it is no longer pinned. This also allows IPFS to become a web where links are permanent (Benet, 2014, p. 7).

A pinned object is an object that the user has chosen to download to the local storage for faster access. Thus, IPFS does not have to keep fetching the object over the network when they want to access it.

Object-level Cryptography

IPFS automatically verifies signatures and can decrypt data with a user-specified chain. Links to encrypted objects are also protected, which will make traversal impossible unless the correct key is supplied.

As a cryptographic function will generate a different hash for the specified object, it is not possible to re-use the objects hash after it has been encrypted. However, if you want to share the object with the old hash, it is possible as long as you decrypt it first and share the decrypted object (Benet, 2014b).

Problems IPFS solves

The following section describes different concerns that plague other technologies and that are solved by IPFS.

- **Malicious attacks**

With the increasing amount of services on the internet, we are seeing an increasing amount of attacks against these systems. The biggest attacks come in the form of Distributed Denial of Service (DDOS) attacks, which is a form of attack that makes use of vulnerable devices to form a network, that is then used to attack a service which in turn will bring it down due to the massive traffic load it brings on the target services network.

On February 28th, 2018, GitHub, a major site used for hosting and sharing projects, suffered from one of the largest DDOS attacks in history. The attack overloaded their network with 1.35 Terabit/second which brought down the site for 5 minutes (Kumar, 2018). Five minutes of downtime might not seem like much, but it is enough to disrupt operations that could result in sites suffering economic damage. Applying the same attack on sites without the resources of GitHub could bring them down for considerably longer durations.

As stated by Amber (2015) "IPFS also addresses security problems that plague our HTTP-based Internet: Content-addressing and content-signing protects IPFS-based sites, making DDoS attacks impossible.". This means that there will be a need to move the web to a more distributed network to prevent these kinds of attacks that are currently employed on the web.

- **Sharing Content**

Sharing content with other people and users without the use of costly hosting hotels has been an issue for multiple years. Most often, content is shared by using third-party websites and applications that do not permanently store content (e.g. Verhoef and Grinfelds (2011)). IPFS solves this problem by allowing users to share content through their own devices if they are online and connected to the IPFS-network. They can share their content with anyone who has the hash and as mentioned, the only requirement is that at least one node has the content that they are trying to share.

- **Availability and Reliability**

The main problem regarding availability and reliability in common-used applications and services are that if they happen to have downtime, the resource that is being accessed might not be available. There are also problems regarding reliability, due to man-in-the-middle attacks and similar attacks, since the resource that is being accessed might be targeted and the user can receive a modified version of the specified file.

Both these aspects are solved in IPFS, due to the use of multiple nodes, replication of resources and hashing of content. As the user will only fetch data with the specific hash requested, if there is an attack that intercepts the transmission and changes the data, the recipient will not receive it at all, as the hash will have been changed.

When a resource is being accessed on IPFS, the user asks for the hash corresponding to the specified resource. If the resource exists on multiple nodes, the user can receive the content from any of these nodes. If one or more nodes fails, the resource can still be accessed if one of more nodes has the specified item. Regarding reliability, when the user requests the resource by hash, IPFS calculates the hash for the resource and verifies it to the specified hash. Thus, whenever content is downloaded, IPFS makes sure that it is the content that was requested.

2.6 System attributes

In this study, the main attributes that are looked further into are scalability, performance and security (Coulouris et al., 2011, p. 527). These are all key features in a distributed file system. Performance, which means how fast the file system can respond to requests and deliver the specified content, is necessary since if there is a long delay, the user becomes frustrated and starts to look for new alternatives.

Because a distributed file system is aimed to be used by many users at once, scalability is also very important, especially for peer-to-peer systems, since one of their key strengths is growing to a massive size and handling multiple users.

Finally, security is a vital aspect when distributing files. It is important that the file system allows for file integrity and that the files stored have not been tampered with. However, achieving this is seldom possible without affecting performance (Xiong, Goryczka, & Sunderam, 2011).

These three system attributes are linked together, and it is generally not possible to have high scalability, performance and security all at once. Therefore, due to IPFS being a relatively new distributed file system, it is interesting to see how these three attributes correlate together and how they affect each other.

3 Problem definition

3.1 Aim

The aim is to evaluate the potential to meet transparency in IPFS with respect to the system attributes, scalability, security and performance.

3.2 Motivation

The motivation for this study is that in the articles we have read, there are no recommendations addressing when IPFS is and is not preferable for use.

Further, during the preliminary research, we found no articles that evaluate how the aspects of transparency, mentioned in section 2.4, are affected in IPFS. Regarding transparency, this subject is interesting since it covers how distributed file systems are perceived by the user in comparison to a local file system, thus interesting in the domain of IPFS.

The aim of transparency is to make certain aspects of the access to the distributed file system invisible to the developer or user, thus important for a distributed file system that aims to appear and operate as a local file system (Coulouris et al., 2011, p. 34).

The system attributes were chosen since they are aspects of quality of service in a distributed system, where reliability was interpreted as scalability (Coulouris et al., 2011, p. 34).

3.3 Research Question

1. What is the potential for transparency in IPFS compared to NFS and ext4?
2. How is the potential transparency affected by increasing the number of IPFS nodes in the cluster?
3. How well does IPFS perform compared to ext4 and NFS with regard to average operation completion times, bandwidth and processor usage?
4. How well the built-in security in IPFS work against attacks, such as man-in-the-middle, DDOS, spoofing etc. compared to ext4 and NFS.

3.4 Objectives

1. Configure an experimental setting for IPFS, NFS and ext4. Perform experiments to measure performance on the mentioned file systems.
2. Configure an experimental setting for IPFS and perform scalability experiments on IPFS.
3. Analyze the security of IPFS and NFS and how well they fare against general security threats.
4. Analyze the collected performance and scalability data to compare how this affects the potential transparency of ext4, NFS and IPFS.
5. Analyze subjective aspects of transparency with a qualitative analysis.

6. Identify and handle threats to the validity of the study.

3.5 Hypothesis

1. Our hypothesis is that the chosen aspects of transparency are perceived as good or better with IPFS compared to NFS. Compared to ext4, the performance of IPFS should be equal or worse.

2. The hypothesis is also that the chosen aspects of transparency are better when more nodes are added to the IPFS cluster

3. The built-in security in IPFS is better than the built-in security of NFS.

Since IPFS is a peer-to-peer file system that is based on BitTorrent technology, where the anecdotal evidence has been that multiple peers downloading the same file increases the performance. This is the reason for the expected hypothesis of IPFS regarding both hypothesis 1 and hypothesis 2.

As discussed in section 2.5, IFPS solves multiple problems, such as malicious attacks. Due to this and since IPFS utilizes secure hashing, we deemed that IPFS should be more secure than NFS.

3.6 Areas of Responsibility

In this study there are certain areas that have been conducted individually as well as areas that have been collaborated on. The main focus of the study is the system attributes, performance, scalability and security. Mattias has the responsibility of the performance aspect of the study which corresponds to objective one. Oscar has the responsibility of the scalability aspect and objective two. Finally, Jimmy has the responsibility of the security aspect and objective three. Further, objective four was performed by Mattias and Oscar and objective five and six were collaborated on by all three.

Each part that corresponds to the system attribute, such as the methodology and results of each attribute has been written by the person who has the responsibility of that area. The remaining areas have all been written collaboratively.

The code used for the simulation of IPFS (deploy.sh and deploy_cluster.sh) was adapted by Oscar. The code was used by both Mattias and Oscar in their experiments. The code adapted for the simulation of IPFS initiates IPFS nodes for each specified machine and used a program called trickle (Eriksen, 2005) to limit the bandwidth of each IPFS node.

4 Methodology

The general methodology for this study was to perform experiments where we can control the environment and variables that could affect the outcome of the experiments. Further, we perform a qualitative analysis where an experiment would yield data that would not be useful, as the results would lead to a probabilistic answer (i.e., completely, partially or not fulfilling the aspects).

There are other methodological strategies that were taken into consideration, but ultimately, they did not suit our needs. Section 0 address why these strategies were not applied in more detail.

4.1 Shared experimental settings for Scalability and Performance

The experimental setting for both scalability and performance was run on the operating system CentOS on machines with Intel Xeon W3550 (Intel, 2009) and 24GB RAM in a non-GUI environment. For all experiments, the software project go-ipfs v0.4.13 (Benet, 2014a) was used for the file transfers. Further, the IPFS version of the IPFS nodes was also v0.4.13.

The experimental configurations for IPFS share the same environmental simulation, where network bandwidth limitations are implemented, and multiple nodes are run on one machine. When selecting the replication distribution, the python 3 library random was used, or more specifically, random sample (Python Software Foundation, 2018).

The implemented limitations are that all nodes are limited to 100 megabits per seconds (100 Mbit/s). This limitation was chosen with the motivation that the Swedish government aims to meet the goal that 95% of residential and company connections should have at least 100 Mbit/s by 2020 (Regeringskansliet, 2016). Thus, these limitations were sensible since most residential and company areas will have 100 Mbit/s within 2 years, as of writing this study. These limitations are implemented by using a tool called trickle (Eriksen, 2005).

The cluster contains multiple nodes, where one or more nodes function as gateways for incoming requests. All nodes in the cluster are fully connected, meaning that all nodes have knowledge of each other and can connect to any node in the cluster. This is a realistic setting for a local area IPFS cluster, such as a cluster deployed at a company, since a company would likely configure a set number of nodes and optimally, these nodes would have knowledge of each other.

4.2 Scalability

This section includes the strategy, method and implementation for the scalability experiments on IPFS.

4.2.1 Chosen Strategy

Experiment

For scalability, the chosen strategy was an experiment in the form of an environmental simulation where multiple instances of IPFS were run on a physical machine, then connecting the virtual machines running IPFS to form a local cluster.

Moreover, a simulation was the preferred method to evaluate the scalability of IPFS since the uncertainty and lack of control of a case study introduces more risks and could yield more uncontrollable variables that could affect the outcome of the study.

Experimental variables

- **Number of nodes**

  The total number of nodes in the current cluster. The configurations are 16, 32, 64 and 128 nodes in the cluster.

- **Degree of replication**

  The proportion of nodes that have replication (i.e., $\frac{1}{8}$ means that one eight of the cluster has replication). The configurations are $\frac{1}{8}, \frac{1}{10}, \frac{1}{12}, \frac{1}{14}$ and $\frac{1}{16}$.

- **Average download times**

  The average time for downloading the given content from the local IPFS cluster

4.2.2 Method Implementation

IPFS

The experiment was conducted with the use of multiple IPFS instances, forming a local IPFS cluster. The simulation starts off with a configuration of the specified replication factor and the smallest cluster size (i.e., cluster size of 16) and increases the cluster size for each trial of downloading files, where one or more nodes have caching turned off. Turning of caching for these gateway nodes was important, since if caching was turned on, these nodes would deliver content from their local storage, thus not querying the cluster. These nodes are used to download the content from the IPFS cluster and evaluate the total download time.

Additionally, the factor of replication was of varying size, this method was based on the method of Guirat and Filali (2013). This was done to measure if replication has an impact on the scalability of IPFS.

The nodes acting as download clients handle multiple concurrent requests, thus simulating a more realistic workload. This yields more data in a shorter amount of time. This allows for more configurations to be run.

The simulation only evaluates the scalability of IPFS, since scaling transparency is relative to itself, i.e. the behavior of the file system when the cluster contains less nodes. Since the system attribute scalability also is measured with behavior and performance of instances with different number nodes, the results can only be evaluated from previous configurations. The results can later be compared to other file systems, but since neither ext4 or NFS have native implementations of replication and scaling, no scalability evaluation was performed on NFS or ext4.

Experiment environment:
To perform the experiment, the software project mentioned in section 4.1 was distributed to a fraction of all nodes in the simulated cluster, not counting the gateway nodes. By randomly distributing the content amongst the nodes, the gateway nodes have multiple nodes to fetch information from, thus allowing the nodes to not only request data from one node.

Evaluation methodology of scaling transparency
The results of the scalability experiment of IPFS were compared to other distributed file systems to determine if IPFS perform better or worse than current competitors of IPFS.

There are not any scalability experiments performed on other distributed file systems in this study, therefore the comparisons are made to related work on other distributed file systems and peer-to-peer systems (Bharambe, 2005; Wang *et al.*, 2013; Mukhopadhyay, Agrawal, Maru, Yedale and Gadekar 2014), such as Ceph and Hadoop (Lay & Dijcks, 2010; Weil, 2007).

Evaluation methodology of replication transparency
Replication transparency was evaluated by collecting data from the scalability simulations, where different cluster sizes and replication factors were used. Furthermore, the data was analyzed to determine if adding more nodes or increased replication factor had any effects on replication transparency (i.e. affecting reliability or performance).

4.2.3 Dependent and independent variables

The chosen variables for the simulation can be seen in section 4.2.1. Average download time was a dependent variable since this variable can be affected by outside variables (e.g., bandwidth limitation). The number of nodes and degree of replication are independent variables, since both variables do not depend on outside variables and are not affected by variables outside of simulation limitations (e.g., hard drive capacity or RAM limitations).

The metric average download time was chosen since it accurately represents normal usage of a distributed file system. When a user tries to access a file, it is more plausible that the user tries to access the entire file instead of only a partition of it.

Due to this, average download time were chosen as the principal indicator metric. The other metric was the total number of nodes in the local IPFS cluster for the current iterations of downloads. This was chosen to measure scalability of a peer-to-peer based system, the number of nodes was a vital measurement for accurately determining scalability.

Finally, the degree of replication was chosen as a metric, since this affects the number of nodes that can deliver the specified content. Since higher replication factor can improve fault tolerance (i.e., one or more nodes can fail, as long as the data is accessible on one or more nodes) due to routing algorithms and redundancy in accessible data (Coulouris et al., 2011, p. 442), the degree of replication is interesting. Another reason was that a higher replication factor allows for more load balancing, theoretically improving performance (Coulouris et al., 2011, p. 431).

4.3 Performance

This chapter describes the implementation details and how the performance experiments are carried out.

4.3.1 Chosen Strategy

Experiment

The methodological strategy for performance is to conduct an experiment. The experiment is an environmental simulation performed on IPFS, NFS (version 4) and ext4. The experiment settings are shared with the scalability experiment and described in section 4.1.

Experimental variables

These are the variables used in the experiment that varies between configurations.

- **File system**

  The file system that is used for the experiment, the three possible configurations are ext4, NFS and IPFS

- **Cache usage**

  If cache is active or inactive during the experiment. As ext4 does not support it, it is always inactive.

- **Replication usage**

  If the files used for the experiments can be found on more than one node. Only IPFS has native support for it, thus it is only present in the IPFS experiment. It is limited to replication on 10 nodes out of 100.

4.3.2 Method Implementation

The Experiment

The experiment in the performance aspect of the study is conducted by analyzing how IPFS compares to ext4 and NFS in terms of the time it takes to complete a set of standard file operations. This is to evaluate how the performance transparency is perceived in IPFS and NFS, compared to ext4.

The method used for the experiment is inspired from the method in the article by Howard *et al.* (1988) in their benchmark test where they perform different file operations.

The experiment in this study performs the same file operations as in Howard *et al.* (1988) except for compilation due to limitations in our experimental environment. Building the project would require multiple steps and would consist of more than the compile file operation.

The four file operations aim to be representative of normal user activities on a distributed file system.

Each experimental configuration consists of running each file operation once and afterwards collect the bandwidth usage in the form of read and write speeds in KB/s.

File operations of the experiment:

The abbreviation in parenthesis after each operation represents the command that is available on CentOS and similar Linux operating systems to execute the file operation (Free Software Foundation, 2017).

1. **Make directory (mkdir)**
   Creates a new directory from the source directory with all subdirectories but without any files

2. **Copy files (cp)**
   Copies every file from the source directory to the new directory

3. **Check status of files (stat)**
   Checks the status on every file in the new directory but not actually reading them

4. **Read all files (cat)**
   Reads every byte of every file in the new directory

After every execution of the four file operations, the newly created directory was removed. This was to allow each trial to execute under the same conditions. From these file operations, data was collected in form of how long it took to complete each operation and what the processor usage was. A configuration on this experiment consists of a combination of three independent variables. These variables are the type of file system, if caching was used and if replication was used.

These independent variables affect the dependent variables measured in the study; Completion time, processor usage and bandwidth usage. Every trial was performed 1000 times to increase the statistical power.

The following tables represent the different combinations of independent variables that are run. Each combination represents one configuration.

Table 1 Ext4 configurations

#	File system	Cache	Replication
1.1	Ext4	Inactive	Inactive

As ext4 does not support either caching or replication only one configuration was performed on this file system.

Table 2 NFS configurations

#	File system	Cache	Replication
2.1	NFS	Inactive	Inactive
2.2	NFS	Active	Inactive

The NFS version used in these experiments does not support replication and thus caching was the only independent variable changing.

Table 3 IPFS configurations

#	File system	Cache	Replication
3.1	IPFS	Inactive	Inactive
3.2	IPFS	Active	Inactive
3.3	IPFS	Inactive	Active
3.4	IPFS	Active	Active

IPFS supports both caching and replication and thus there are four trials performed.

The NFS setting consists of one machine acting as a server while the other acts as a client, mounting a remote directory from the server machine. Finally, the IPFS cluster consists of 100 isolated nodes that are simulated on two different machines. To simulate network bandwidth restrictions for NFS, a program called Comcast (Treat, 2017) was used, capping bandwidth usage to 100 Mbit/s. The IPFS trials are performed with the settings described in section 4.1.

Performance Transparency

The data from the performance tests was used to find out how the performance transparency in NFS and IPFS is likely to be perceived. The ext4 results are used as a baseline due to it being a local file system. Further, to be able to objectively draw conclusions from the IPFS data it was also compared to NFS as it is another commonly-used distributed file system.

As transparency is a perceived feeling of how a distributed file system feels like it is a local, it cannot be measured directly. Instead, conclusions are drawn from the performance results and compared.

According to Neilsen (2010) research has been performed on response times of websites in Nielsen (1997) and the research concluded that the response time limits are up to 10 seconds. A delay of more than 10 seconds caused frustration to the user and made them leave the site. Important to note that waiting 10 seconds for a web page to load is extremely long. Their research stated that average users tend to get frustrated if it takes more than one second for a web page to load.

The research from Neilsen (2010) inspired how to conduct the analysis of how the performance transparency was perceived in IPFS and NFS. The main difference is that Neilsen (2010) measured time limits while in this study we measure the time it takes to complete an operation on the local file system as a baseline and compare the time it takes to complete the operation with NFS and IPFS.

For example, if it takes one second to execute the operation on ext4, then it is expected to take roughly the same amount of time on the other file systems. If it takes 10 or more seconds to execute on NFS or IPFS compared to the baseline, that is established by executing the same operation on ext4, then we deem it as unacceptable and has too much of an impact on the performance transparency.

However, our assumptions are that if the baseline operation already takes a long time to execute, such as copying a large file, the time difference does not matter as much, as it was already expected to have a long delay. This in turn would not impact the perceived transparency to the same extent.

Replication Transparency
Analyzing the replication transparency is performed by adding the same content on specific nodes when executing the IPFS experiments.

By distributing the same content on multiple nodes, replication transparency can be analyzed. Selecting what nodes would have replication was done by using the same library as described in section 4.1. This was done by verifying that the content downloaded is the same as the one requested and this was done automatically by IPFS. The results are compared to the completion times of IPFS without any replication to see if replication affects performance. If the times greatly differ this could mean that multiple replication affect the performance.

Replication was only measured on IPFS as ext4 and NFS do not support native replication of files.

4.3.3 Dependent variables

The dependent variables used for analyzing performance were average completion time and processor usage of each operation and average bandwidth for all operations. The motivation for these three is that they are measurable. As well as in Howard *et al.* (1988), processor usage and completion time of operations were used. Bandwidth was chosen because it represents how much data is transferred over a given time. (TechTerms, 2012)

4.4 Security

This section about security describes the common security threats and how they are analyzed in NFS and IPFS.

4.4.1 Chosen Strategy

Qualitative Analysis

The security aspect of the study was performed by conducting a qualitative analysis by looking at previous research that has been done on the security methods for the various systems, followed by a qualitative analysis of the findings.

4.4.2 Method Implementation

There are several risks that are associated with distributed file systems; These risks consist of Leakage, which includes Eavesdropping, Tampering which includes Masquerading, Message Tampering and Replaying, and Vandalism which includes Denial of Service.

Each risk is described in detail in Coulouris *et al.* (2011, p. 466-467), and these are risks associated with distributed systems, not only distributed file systems.

Leakage – Leakage refers to the risk of having unauthorized users gaining access to classified information via a sort of leak in the organization, for example a whistle blower or a data leak.

Tampering – Tampering is the risk of having an unauthorized party editing/altering information that said party should not be able to edit or alter.

Vandalism – Vandalism is when an unauthorized party actively interferes with the operation of a system without any gain to themselves.

Eavesdropping – Eavesdropping is the unauthorized copying of messages being sent between two parties.

Masquerading – Masquerading is when a malicious entity is assuming the identity of another principal to send messages with.

Message Tampering – This is the act of intercepting messages being sent, and then altering them before sending them on to their destination. Man-in-the-middle attacks are a form of message tampering that intercept the first key exchange, to then replace with keys of their own so that they can decrypt messages being sent for the duration of that session.

Replaying – Storing the intercepted messages to use at a later date, for example could be used for replaying messages for bank transactions. These intercepted messages work well even for authenticated and encrypted messages.

Denial of Service – Flooding a channel with messages in order to overload it and take it down and prevent others from accessing it.

Distributed Denial of Service – Distributed Denial of Service is very similar to a regular Denial of Service. The big difference between the two is the magnitude of the attack possible. A Distributed Denial of Service attack makes use of entire networks of compromised devices to flood channels with much more traffic than is possible with a regular Denial of Service.

IPFS and NFS
A qualitative analysis on both IPFS and NFS is performed, and then compared to the baseline, and graded in Table 4, on how they handle the risks, or if they possibly make it worse.

Analysis of Security related transparency
The following transparencies are important when it comes to securing a system. When securing a system, it should not add any more steps than necessary for the user. It should also not cause any issues in operation, such as lowering performance to the point where it is noticeable to the user.

Therefore, the following transparencies will be analyzed in regard to how they are currently handled in both NFS and IPFS, and any planned solutions to improve it.

- Access Transparency
- Migration Transparency

Access Transparency
In regard to security this means that the resource in question should be able to be accessed even while being secured, without having to add extra steps for the users. Ideally the user should not have to be aware that the item they are accessing is secured at all.

Migration Transparency
Migration Transparency in regard to security means that users should be able to move around the system and be able to move resources that they have permissions for, without being negatively affected by the chosen security implementations.

4.5 Qualitative analysis of subjective aspects of transparency

The following transparency aspects differ from the previously mentioned aspects in the way that they cannot be objectively measured in the same way that the earlier aspects can. Instead they are measured in a subjective way by conducting a qualitative analysis.

- Access transparency
- Location transparency
- Failure transparency
- Migration transparency

The qualitative analysis was performed by reviewing what defines the transparencies according to Coulouris *et al.* (2011, p. 23). With this information in mind, we analyzed the documentation of IPFS and NFS, and how they met the criteria of these transparencies.

4.5.1 Access Transparency

The access transparency was analyzed by considering how you access the file system in ext4, NFS and IPFS. Ideally you should not have to perform any extra steps to access the file system such as login onto an FTP connection first.

4.5.2 Location Transparency

When analyzing the location transparency, it was considered how the location of files matter between the file systems. If a user wants a file, it should only have to provide the file name to access it. Not having to specify what server or node that holds the file.

4.5.3 Failure Transparency

This was analyzed by considering how IPFS handles the ability to receive files if suddenly a node that has part of the files crashes. Likewise, in NFS it was analyzed to see what happens if a server that holds the data suddenly crashes, and a backup server handles the connection. Does the user notice this change at all? For this aspect it was not possible to analyze it from the ext4 point of view, due to it being a local file system only.

4.5.4 Migration Transparency

This transparency was analyzed by considering if it was possible for resources in the distributed file system to be moved to a different location without the path changing, and the user should not be aware that the contents has migrated. The reallocation of the resource should not affect operations on the resource at all.

4.6 Handling of validity threats

In this chapter, we describe possible threats to the validity of the study and how we handle them.

4.6.1 Conclusion Validity

Low statistical power

There is a risk that the amount of data is not enough to draw statistical conclusions from. To combat this threat when performing our scalability and performance experiments, we run more than 1000 trials per test we conduct. This way we reduce the possibility that outliers affect our results giving us more accurate data (Wohlin *et.al.,* 2012, p. 104).

Fishing

Fishing is a threat where the person conducting the research may attempt to alter the study in order for the study to yield a specific result. This becomes a threat due to the study being no longer independent and the results may be due to the influence of the researcher. An example of how this could be applied in this study is that we only show the "good" result of IPFS in order for it to appear a lot better than it might be.

The way that we handle this threat is by presenting all the collected data from the experiments, as well as describing the steps performed allowing the study to be recreated to validate the results. Further, we combat this threat by objectively comparing the IPFS data with another distributed file system and draw our conclusions from this. Furthermore, we only make subjective evaluations on the four aspects of transparency that we deem to be subjective (Wohlin et al., 2012, p. 104).

Reliability of measures

There exists a risk that the simulation environment is inaccurate. This is handled to the best of our abilities, since the software used for simulating the environment does not create simulated IPFS nodes, but instead it creates actual nodes on the same system where all nodes are limited by third-party software. Since there are multiple nodes working concurrently on the same system, there is a possibility that the environment does not behave as they would in a real world setting (Wohlin et al., 2012, p. 105).

Random irrelevancies in experimental setting

In this study we are running multiple IPFS nodes on two host machines. This may result in problems that could influence the results of the study. There may be additional overhead due to the amount of context switching that occurs when switching between the different IPFS nodes. However, important to note is that while the environment surrounding the IPFS nodes is simulated, the actual nodes themselves are not. They are running genuine IPFS software and because of this, they perform and act as if they would reside on an individual machine.

In an optimal situation, each IPFS node would be run on a unique machine. This would reduce the risk that random updates on the host machine could cause discrepancies or similar problems that originate from having multiple nodes on the same machine. However, we did not have access to enough machines to simulate a complete cluster with the given size.

To handle this threat, to the best of our abilities, we disable random updates as a security measure as to make sure that trials do not have different versions of tools. Disabling random updates also ensures that less unwanted background actions are executed that could potentially skew the data (Wohlin et al., 2012, p. 106).

Random heterogeneity of instruments

As the experiments are executed on the same operating system and the same hardware it is completely homogenous. This results in very low heterogeneity in the experiment. Low heterogeneity in an experimental setting is a good thing as it reduces the risk that the results are due to the difference between the systems.

The way we handle this threat is by acknowledging the fact that it may affect results, but handling this threat is not feasible or in scope for this study (Wohlin et al., 2012, p. 106)

4.6.2 External and Internal Validity

Due to this study mainly consisting of experiments and qualitative analysis' we do not have any test subjects. This means that many threats regarding test subjects simply do not exist in this study. As we run our experiments on machines we can run our tests multiple times without worrying about the tests are being affected by previous iterations, which is a major concern when using test subjects.

4.6.3 Construct Validity

Inadequate preoperational explication of constructs

This threat means that constructs are not properly defined before they are translated into measures. For example, if we only were to compare IPFS and NFS without clearly defining in what ways we compare them, we would not be able to say that either is "better". We handle this threat by clearly stating, in each of the three system attribute chapters, how the comparisons of the file systems are carried out (Wohlin et al., 2012, p. 108).

Confounding constructs and levels of constructs

Due to limited knowledge in how to use IPFS, this threat may affect the results of our experiments because of how we set up our experimental simulation (Wohlin et al., 2012, p. 109). We use the official documentation from IPFS to perform specific operations and we do not use third party implementations. This limits the possibility that our experiments use IPFS in an unsupported way that may skew results.

4.7 Alternative methodological strategies

While previous mentioned strategies were chosen for this study, there are more strategies that were not implemented, for various reasons listed below:

Case Study

Case studies require real-life scenarios. This leads to less control over variables compared to an experiment but also requires more hardware. Due to this, a case study is not a feasible strategy for this study. (Berndtsson, Hansson, Olsson and Lundell 2008, p. 62)

Survey and Interview

Conducting a survey on IPFS would be difficult since the file system is relatively new and not very well known. Thus, it would be difficult to get any meaningful data from the survey since the domain of this study is a niche aspect of distributed file systems.

A survey would yield subjective answers, thus affecting the validity of the results. Likewise, the same problems arise if we were to perform an interview. We do not want subjective answers because depending on subjects of the interview, they may experience the systems differently (Berndtsson et al., 2008, p. 63). It would also be difficult to verify the answers and results from a survey or interview, especially considering the topic of this study.

Action Research

An action research is very similar to a case study, with the exception that the researcher themselves will partake in the experiment. This is not a feasible strategy since it has the same problems as a case study (Wohlin et al., 2012, p. 56).

Literature analysis

A literature analysis takes a lot more time to conduct than the time frame given for this study. Moreover, it also requires large quantities of previous research papers on the chosen subjects. Due to IPFS being a relatively new distributed file system, there is not enough research to successfully conduct a literature analysis (Berndtsson et al., 2008, p. 58).

Meta-analysis

To perform a meta-analysis, there must be previous studies done in order to get quantitative data from several cases. Since IPFS is a new distributed file system, not enough research has been performed to successfully conduct a meta-analysis (Wohlin et al., 2012, p. 51).

5 Related work

In this chapter, the scientific contributions from this study are discussed.

5.1 Scalability

One of our contributions for distributed file systems was the method implementation for peer-to-peer based distributed file systems, or more specific, for IPFS. One difference between the implemented method in this paper and methods in earlier research, such as Wang *et al.* (2013), was the usage of bandwidth shaping per virtual node. As previously mentioned, this means that each instance of IPFS has limited bandwidth. This is to simulate network limitations that exists in a real-world setting, thus emulating a real-world scenario.

Due to these implementations, each node can be restricted to a set bandwidth, thus multiple nodes can function on the same host. This enables experiments without the need for multiple physical machines, increasing the feasibility for others to conduct similar research. Another contribution is that this study uses greater cluster sizes compared to Gudu, Hardt and Streit (2014) and Wang *et al.* (2013), which is a more realistic setting for IPFS.

This research also contributes to IPFS, since there are few research papers published that discusses or evaluates the scalability of IPFS. Due to this, this paper can be a stepping stone for IPFS and a reference for others whom are interested of IPFS.

5.2 Performance

This research contributes by using similar methods performed in previous research to measure performance, for example in Howard *et al.* (1988), where performance was measured between Andrew File System (AFS) and NFS. Our research uses similar methods to measure the performance of IPFS instead of AFS.

We also analyze how the transparency of the file systems was perceived. This is interesting for private users or corporations who may want to start using IPFS and can follow this study as a basis to evaluate if the file system should be used or not, mainly in terms of the performance of IPFS.

5.3 Replication

In our performance and scalability research, we measured replication with varying degree. Both showed that replication appeared to have a negative effect on scalability and performance, with higher degree of replication affecting the scalability and performance more. These findings contradict the results we found in previous research on replication in Bharambe (2005) and Gudu, Hardt and Streit (2014), where the latter research found that Ceph (a peer-to-peer based distributed file system) scales almost linearly when adding more nodes with replication.

Our findings are interesting for the researchers and developers of IPFS, since our findings can help the improvement and help the development of IPFS.

5.4 Security

The largest contribution of security is how it allowed the reader to determine which of the two file systems are the correct choice for them. If they want a local smaller and more secure file system or if

they want something more suitable for a larger network over the internet, but without the added security (as of right now) that NFS offers.

In the analysis of security on NFS and IPFS, it was shown that NFS is still the better option as it IPFS is right now, especially if it should be able to host data on an internal network for a private organization that needs their data to be secure.

6 Evaluation

This chapter includes the results from the experiments and qualitative analyses performed on Scalability, Performance and Security.

6.1 Scalability

This section discusses and analyses the results from the conducted scalability experiment and tries to answer the related research question. There was also analysis performed on the transparency aspects related to the scalability experiment, such as scaling transparency and replication transparency and can be found in section 6.1.2.

6.1.1 Results

Results from the scalability experiments are visualized as point plots, boxplots, bar graphs and line graphs. All graphs have a y-axis of download times, while x-axis are cluster size or replication factor. There are also figures for describing spread, mean and standard deviation for each configuration. Graphs and tables can be found Appendix A -, Appendix O - and Appendix P -.

Figure 1 visualizes the download times for the software project mentioned in section 4.1 with the spread of each configuration. The download times are split into the different replications factors run during the experiment to visualize the effects replication factor has on download times.

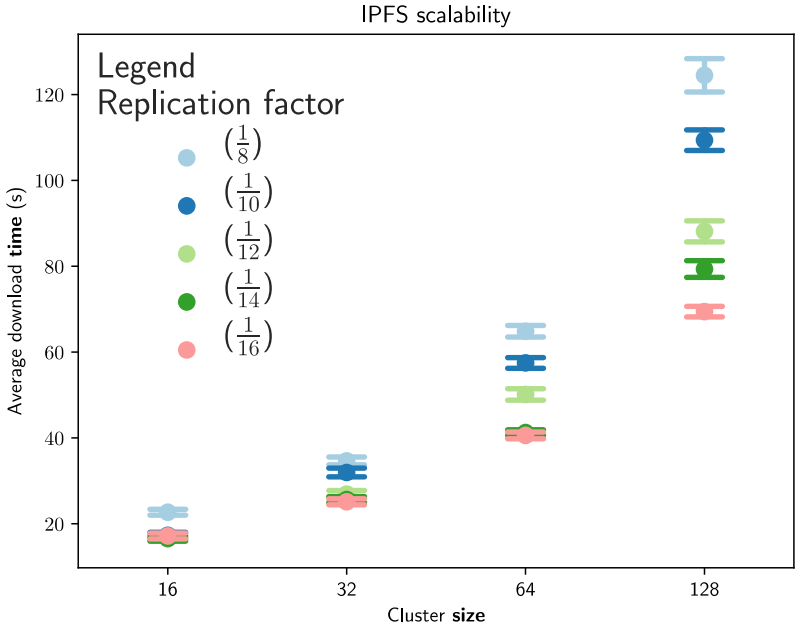

Figure 1 Average download time with varying cluster sizes and replication factors

6.1.2 Analysis

As the results from these experiments show, the replication factor of the cluster greatly affects average download times for IPFS, more so than cluster size. When comparing a replication factor of $\frac{1}{16}$ with a cluster size of 128 with a replication factor of $\frac{1}{8}$ but a cluster size of 64, there is little difference in download times. When comparing these replication factors with the same cluster size (e.g. 128), there

is a significant difference in average download times. These findings greatly contradict the findings of Bharambe (2005), where mean download times improved with more uploaders (i.e., higher replication). The results from our study seems to imply that if there is a higher replication factor, the mean download times would increase. Thus, since our results greatly differ from earlier research, we speculate that there might be a problem with either our environmental simulation, or IPFS itself. Due to these concerns, a smaller case study was performed and discussed in section 7.6.

Results with high replication factor

As seen in Figure 1, simulations run with a high replication factor (e.g., $\frac{1}{8}$) performs worse than simulations run with a low replication factor (e.g., $\frac{1}{16}$). Comparing scaling performance of the high replication factor, the results indicate that replication factor has a great impact on the overall performance of IPFS. This can be seen further in Appendix O -, analyzing the different download times for the high replication factor, as the results scale worse compared to a lower replication factor. When comparing the cluster sizes 16 and 32 for the replication factor $\frac{1}{8}$, mean download times are 22.69 and 34.66 seconds respectively, an increase of 53%. Comparing this increase to a cluster size change from 32 to 64, download times went from 34.66 seconds to 64.85 seconds, an increase of 87%.

When comparing the same cluster sizes, but with a replication factor of $\frac{1}{16}$, mean download times where 17.16, 25.65 and 40.59 for cluster sizes 16, 32 and 64 respectively. The increase mean download times were 49% and 58%. When increasing from cluster size of 32 to a cluster size of 64, the replication factor $\frac{1}{8}$ saw an increase of 87% while the replication factor $\frac{1}{16}$ saw an increase of 58%. These results indicate that the replication factor is the principal factor for IPFS and affects scalability more so than cluster size.

Difference in nodes with replications

As mentioned earlier, there was a huge impact on performance when increasing replication factor of the cluster, more so than increasing size. As shown in Figure 8 a replication factor of $\frac{1}{8}$ substantially increases download times, while a replication factor of $\frac{1}{16}$ does not affect download times to the same degree. This can be further seen in Figure 7, where the number of nodes with replication was halved when comparing replication factor $\frac{1}{16}$ to $\frac{1}{8}$, the download times are almost halved as well.

Moreover, when there was a difference in cluster size, there was little effect on download time, as seen in Figure 11. The difference between a cluster size of 64 and 128 with same number of nodes with replication, was roughly 5 seconds. Comparing these results to the difference between those seen in Figure 8, the conclusion that cluster sizes are not the principal factor can be drawn.

Scaling transparency

When adding more nodes to IPFS, scaling transparency was not affected. There were no changes to be done to algorithms or the structure of the system. Adding more nodes to the cluster increases download times to a small extent, depending on cluster size, while adding more nodes with replication greatly increases average download times.

Some questions can be drawn from these results. Is IPFS meant to be used in the way the experiments were constructed or if there is a problem with IPFS? Either way, there are some surprising aspects regardless of the answer of these questions.

Replication and performance transparency
As seen in section 2.4, replication transparency states "*enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers"*. While IPFS does allow for multiple instances to be used, these instances do not increase performance. It does increase reliability, since IPFS allows for the resource to be fetched from any of these instances, but it negatively effects performance.

The replication has less impact when less instances share the resource, but increasing the replication increases the average download times. Thus, adding more nodes with replication to the cluster does not improve replication transparency.

Performance transparency is greatly affected by increased replication, while increased cluster size does not affect performance transparency as much.

6.1.3 Conclusion
According to hypothesis 2, it was believed that the chosen aspects of transparency would be better compared to clusters containing less nodes. These aspects were scaling and replication transparency. Scaling transparency was the only aspect that was left unaffected by increased cluster size and replication factor, while replication transparency was affected negatively.

Scaling transparency was not affected since the underlying system did not change when changing the independent variables. Replication transparency, on the other hand, was affected by changes in the independent variables. The performance was greatly affected by the increase of the replication factor, while only a small change was noticed when increasing the cluster size.

6.2 Performance

This section describes the results from the performance experiments and what the graphs listed below represent. These results are compared and analyzed to see how the aspects of transparency in NFS and IPFS are perceived. There is a conclusion summarizing what these results mean for our study and our original hypothesis as a whole.

6.2.1 Results

The results for the performance trials are visualized as boxplots. The graphs in this chapter are the completion time for the copy operation. Only one operation is shown in the report and the reason for this is that all four operations (Appendix B - to Appendix K -) were relatively close in completion time and showing all operations would be redundant.

In Figure 2 the graph shows the completion time for the copy operation on the three file systems, ext4, NFS and IPFS. With combinations of cache turned on or off. In Figure 3 it instead only compares the copy operation between IPFS, this time with combinations of cache and replication. In both figures, true means activated and false deactivated.

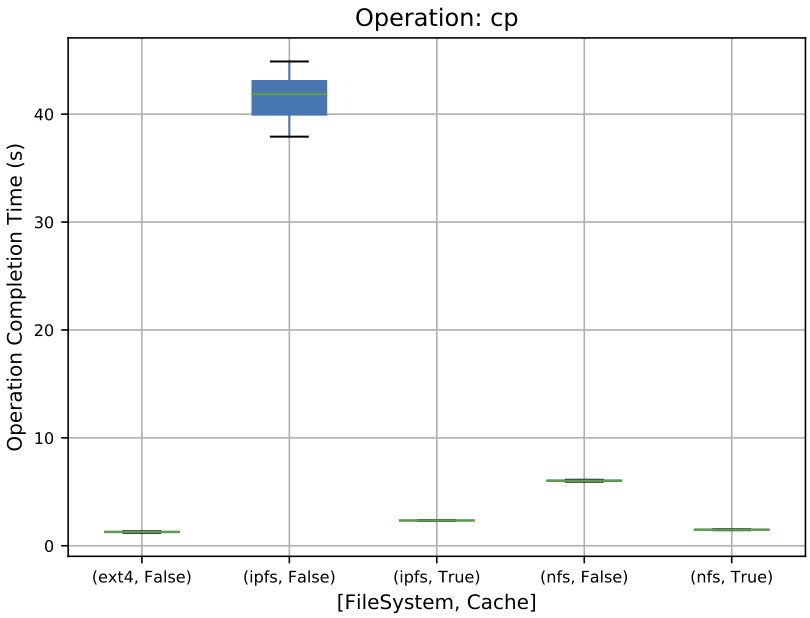

Figure 2 Completion time in seconds for Copy with file systems and cache

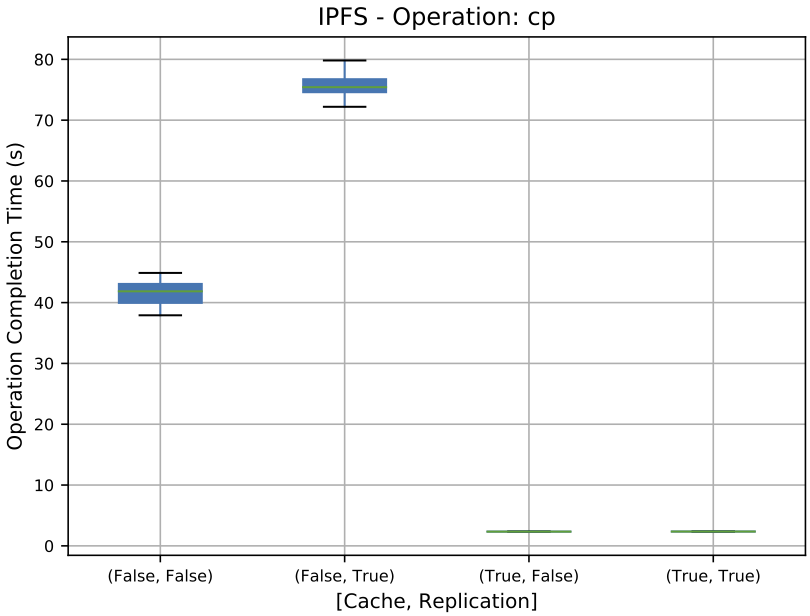

Figure 3 Completion time in seconds for Copy operation on IPFS with combinations of cache and replication

The remaining operations as well as the graphs for bandwidth and processor usage are shown in the appendix from Figure 12 to Figure 34. Complete list of the experiment data that was collected to create the graphs are in Figure 32 to Figure 34.

6.2.2 Analysis

When analyzing the results, the copy command will be analyzed due to as stated, all four operations resulting in almost the same time, with copy being the slowest. The values measured are the median for completion time, bandwidth usage and processor usage.

Results without cache

There are trends that are visible when analyzing the results. The common theme in the overall results was that without cache, the operations for ext4 and NFS do not differ greatly in completion time and are all below 10 seconds in completion time. However, IPFS, for all four operations take upward 40 seconds or more to complete.

The median for the copy operation in ext4 takes 1.28 seconds, in NFS it takes 6.03 seconds and in IPFS the same operation takes 41.85 seconds. The processor usage median for copy in ext4 was 12%, NFS was 4.8% and for IPFS without cache it was 37%. Interestingly the processor usage is lower in NFS than it was in ext4. Something to note though is that only the client side of NFS was measured and the server aspect could have a part in why it was lower on NFS compared to ext4.

The bandwidth read usage for ext4 for all four operations was surprisingly 0 Kbit/s. However, this value is not correct, but due to errors in the measurement tools for bandwidth it would not properly display “correct” results. These errors were that the OS used in the experiments did not correctly report the read sectors of the hard drive and updated these periodically. However, it correctly reported the write sector. The ext4 read results should end up at roughly the same as write as some operations both write and read at the same time, while some only read. The overall bandwidth for ext4 is limited to the performance of the hard drive, thus read and write results should be similar. The bandwidth

read usage in NFS was 669.28 Kbit/s and for IPFS it went up to 1565.62 Kbit/s. The write usage lists roughly the same results apart from ext4 in which it reports 22 Kbit/s.

What can be seen from these results was that ext4 was the fastest, not surprisingly, with NFS slightly slower. IPFS, however, was many times slower with a lot higher bandwidth usage.

Results with cache

With the cache activated, the completion times for both IPFS and NFS on the copy operation are faster. NFS decreases to 1.49 seconds from 6.03 seconds and IPFS decreases to 2.34 seconds from 41.85 seconds, a huge improvement. The processor usage for IPFS goes from 37.15% to 15.90% and for NFS it goes from 4.8% to 12.30%, increasing in processor usage. This can be explained by NFS having to look up if it has something in the cache instead of directly accessing the mount point, resulting in higher processor usage.

The bandwidth read and write usage go from 1565.62 Kbit/s and 961.89 Kbit/s to 0.14 Kbit/s and 0.51 Kbit/s respectively, resulting in greatly reduced bandwidth usage for IPFS. For NFS it goes from 659.83 Kbit/s and 601.56 Kbit/s to 24.10 Kbit/s and 23.90 Kbit/s which also reduces the usage a lot. Something to note though is that only the network bandwidth usage was measured for NFS and IPFS and not the disk usage as in ext4.

The conclusions that was drawn from these results was that cache usage greatly improves performance and should be used whenever possible.

IPFS replication comparison

Activating replication causes the very slow results of IPFS to be even slower without cache. Comparing the copy operation yet again results in completion time without replication go from 41.85 seconds to 75.42 seconds with replication. Almost doubling the time to complete.

The processor usage does not increase as much, from 37.15% to 44.65%. Finally, the bandwidth usage increases from 1565.62 and 961.89 read and write Kbit/s to 4023.28 and 1541.34 Kbit/s resulting in a large increase in read usage and a slightly higher write usage. This was with an IPFS cluster size of 100 and replication on 10 nodes. With cache activated, replication does not matter as the node can retrieve the data directly from the cache.

The conclusion that are drawn from these results, is that with our simulation environment, the replication greatly impacts performance in a negative way for IPFS, almost doubling completion time and bandwidth usage, with no greater effect on processor usage.

Performance Transparency

What effect does this have on the performance transparency of IPFS? As stated in section 2.2, the goal for the file system is to feel local and to measure this the results from IPFS is compared to ext4.

Referring back to Neilsen (2010), the upper limit of load times on websites was 10 seconds. Thus, for this example the ext4 copy operation took 1.28 seconds, on NFS it took 6.03 seconds and on IPFS 41.85 seconds. NFS manages to stay within 10 seconds and the user will probably feel like it is okay. However, IPFS takes 41.85 seconds to execute the very same command that took 1.28 on ext4. This is a big problem and the user will probably be frustrated and wonder if the file system is still even executing.

The performance transparency fails for IPFS due to the extreme time difference and causes it to not feel like a local file system.

Replication Transparency
The replication transparency states "*enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.*"

In IPFS multiple replicas do not show up as multiple copies for the user. However, it does not appear to increase performance in this case. The time to complete the IPFS operations with replication almost doubles. Thus, the performance aspect of the transparency fails.

6.2.3 Conclusion
According to hypothesis 1, the aspects of transparency are experienced as good or better with IPFS compared to NFS and compared to ext4 it is as good or worse. However, the conclusions from our results point to that the performance transparency does not confirm our hypothesis, IPFS performs worse than NFS and ext4 as it takes too long to execute operations. As it is a peer-to-peer system it does not perform better with files replicated on multiple nodes either as it almost doubles the execution time with 10 replicas on 100 nodes.

There are a couple of reasons as to why IPFS could be slower than expected. One of these could be issues in our environmental simulation that causes IPFS to be slow. It is also possible that there are underlying issues with IPFS or simply that it is not intended to be used in this way. As for the replication aspect, replication appears to worsen the performance even more, however, more specific testing would have to be carried out to determine the exact cause of the performance issues.

6.3 Security

The following section will detail the results and contain the analysis of the security research of the file systems in question. It will try to answer the research question of how well IPFS will handle threats that are common for distributed file systems and compare it to NFS, so that the reader could make an informed decision depending on their needs.

6.3.1 Results

According to Anderson (2008) and Coulouris *et al.* (2011), security is an integral part to distributed systems. Their research has shown that there are inherent weaknesses to a network that needs to be solved for a distributed system to be useful.

These risks consist of Leakage, which includes Eavesdropping. Tampering which includes Masquerading, Evading, Message Tampering and Replaying, and Vandalism which includes Denial of Service.

The criteria for evaluating how well each file system handles or do not handle the risks are in low, medium, and high.

Low means that the file system has none or the support for it is bad.

Medium means that there are third-party solutions that require more systems to work.

High means that it has either implicit support due to architecture or software, or explicit where it is natively supported but requires it to be turned on in settings.

Table 4 Grading of security aspects

File system / **Risk**	**NFS**	**IPFS**
Leakage	High	Low
Eavesdropping	High	Medium
Tampering	High	Medium
Masquerading	Medium	Medium
Evading	High	Low
Message Tampering	Medium	High
Replaying	High	Low
Vandalism	Low	Medium
Denial of Service	Low	High

Leakage

NFS

To prevent the threat of having data leaked to parties that should not be privy to the data, NFS has some built-in security measures. One method is that it employs Access Control Lists via three different methods, Lipkey, SPKM-3 and Kerberos V5 (Callaghan, Robinson, & Thurlow, 2003, p. 25), the most common one being Kerberos (Coulouris *et al.*, 2011, p. 544). By using Kerberos, it is possible force authentication of users to both files and directories, preventing unauthorized access and impostor attacks. One other method that NFS employs is the encryption of communication channels. This is to prevent someone being able to listen in on the conversation taking place between the client and server. This is important when dealing with sensitive data, such as transaction handling or personal information via email (Coulouris et al., 2011, p. 469).

IPFS

IPFS allows for encrypted communication channels, but as mentioned by (Whyrusleeping, 2017), it does not currently have any other measures for security. Files cannot be encrypted, nor is it possible to restrict access with Access Control Lists. As it is, IPFS is quite prone to have private data leaked, as everyone on the IPFS network will have access to any files currently shared, unless the file itself is protected via other means.

Tampering

NFS

With the security measures NFS has built-in with Kerberos it is possible to restrict access and prevent tampering of any objects stored in the server.

IPFS

To prevent tampering of files, IPFS makes use of cryptographic hashes that is based on an objects data for each object stored on the network. This cryptographic hash makes it so that if a file has been changed that new edited file will have a new cryptographic hash, which means that any links to the old objects will always lead to the old, original object (Benet, 2014b, p. 2).

Vandalism

NFS

Vandalism is an attack on the system itself, not on the properties of a file system. As such, NFS makes use of the underlying operating system to protect it from vandalism by any malicious entities.

IPFS

IPFS, like NFS, is a file system that runs on top of the host operating system and file system. As such it also handles the threat of vandalism in the same way.

Eavesdropping

NFS

As described by (Coulouris et al., 2011, p. 474) NFS makes use of encrypted channels for communication. The methods for encrypting the communication are public key-pair algorithm that will make it so only those in possession of the key to the channel will be able to read the messages. This is part of the Kerberos part of NFS.

IPFS

As mentioned by (Whyrusleeping, 2017, p. 4), IPFS does not have content encryption. Only encryption of the channel itself. However since the channel is encrypted by the a similar public key-pair algorithm as NFS to make it so only the parties with said key can partake. (Benet, 2014b, p. 3)

Masquerading

NFS

As the Access Control Lists in NFS makes use of a unique ID (Coulouris et al., 2011, p. 544) for access, and not their actual name, which is only for users, it is hard to make it look like someone else did something harmful as one could look for the ID of whatever user did it.

IPFS

Each node on an IPFS network will get their own unique NodeID which is a cryptographic hash of a public-key created with S/Kademlia static crypto puzzle. It is possible for a user to change identity by shutting down their Node and then restarting it. To prevent this, IPFS makes use of a system of incentives to make people more likely to not renew their NodeID all the time (Benet, 2014b, p. 3).

Evading

During the qualitative analysis performed on the security aspect, we managed to find a different kind of masquerading which is commonly referred to as evading. This is the act of changing the client/user identity to a completely new identity, instead of pretending to be an already existing user.

NFS

This used to be a problem in earlier versions of NFS where it was possible to change ID locally and affect what you had access to remotely, this however is fixed in recent versions and thus this is no longer possible.

IPFS

With IPFS, since you can reset your node and generate a completely new NodeID as part of the features of IPFS, there is no protection against people shedding their old identity and creating a new one and pretending to be a new member of the network. What IPFS does to combat this happening is their built-in system for older and trusted nodes to accrue certain benefits, which are lost if a new NodeID has been generated.

Message Tampering

NFS

By encrypting the channel for communication (Coulouris et al., 2011, p. 474) as well as encrypting the object itself and controlling the access with Kerberos, NFS makes it difficult for someone to tamper with a message that is in transit.

IPFS

By encrypting the channel it is possible to prevent malicious entities from listening in. However, the messages being sent are not encrypted, which means that it would be possible for a man-in-the-middle attack to intercept a message, change/read it and send it back on its way. However, since IPFS makes use of cryptographic hashes, it only makes it possible to read the messages, as any changes to it would change the cryptographic hash and thus the receiver would not receive it.

Replaying

NFS

Kerberos, the security mechanism that NFS uses employs the concept of challenges whenever someone is trying to do something. A challenge is a way to prevent replay attacks by requiring you to authenticate yourself in some way every time an action is being taken where sensitive data is involved, for example, you can browse your account at an internet bank without having to authenticate yourself once you are in. But as soon as you want to move money, you will be challenged and must prove that you are who you say you are.

IPFS

IPFS has no mechanisms to prevent replay attacks in its current state.

(Distributed) Denial of Service

NFS

NFS is highly susceptible to these kinds of attacks due to it being a Client/Server distributed file system. This means that there are a set number of units hosting the content somewhere, which means that it is easy to overload, especially with a distributed version of the attack. As we saw with the attack on GitHub (Kumar, 2018), which uses a CDN (Content Delivery Network, a cluster of servers that are load balanced) with a client/server structure. It was possible to bring it down, even if only for a couple of minutes.

IPFS

The biggest benefit for peer-to-peer architecture is that it will be nearly impossible to bring down services making use of this architecture because it will be spread out on hundreds if not thousands of nodes. As Amber (2015) mentions, a peer-to-peer architecture will be needed eventually because of how the internet has developed into a place of services that people gather around and are now becoming vulnerable to attacks that aim to deny service to them. IPFS could also be used to detect and prevent incoming DDOS attacks as mentioned in Rodrigues, Bocek and Stiller (2017).

6.3.2 Analysis

IPFS in its current shape is severely lacking in regard to security. While it does have secure channels right now. It is not enough for IPFS to be used on the internet, as the content itself should be able to be protected via means such as Access Control Lists or encryption.

When adding security to a system there is an inherent risk of making the software slower or harder for the user. To prevent this happening, there are discussions and plans Whyrusleeping (2017) for how to implement security features such as ACL or encryption of files on IPFS in the future. As these things are already common in NFS without making the user be noticeably affected by it.

In the proposed IPFS design, it is possible to have a parent object encrypted under one key, and the children with another, or not all, which will secure shared links to shared objects (Benet, 2014, p. 7).

There is also a recent discussion regarding Merkle trees and how they could be used as an attack vector by employing pre-image attacks on it. In Flawed (2018) they talk about using a second pre-image attack against a Merkle tree. Fortunately, IPFS uses an implementation of a Merkle DAG, as described in 2.5 Merkle DAG, that does not make this kind of attack possible.

As the results from the small-scale analysis in section 7.6 show, IPFS suffers negatively when replication is increased over the network. This vulnerability found in IPFS could be used to

maliciously attack the network by creating a form of Denial of Service as described in 7.9 that is only possible due to the vulnerability. As a peer-to-peer network, IPFS should be implicitly protected from such attacks, but it is not. As a result, it might mean that NFS with a third-party option such as pNFS (Streicher, 2008),which allows for replication, would be a better choice.

6.3.3 Conclusion

If you want a file system for your intranet for hosting your companies classified files, NFS with its various security mechanisms is the only viable choice in this scenario. As an intranet server should not be open outwards, the threat of attacks like Denial of Service is not as high, which means that secure access is more important, to make sure that someone who should not be able to read it cannot read it. Which makes IPFS right now a big no-no as it has no way of controlling access to the content on the network. Everyone can access everything as soon as they join it, which means classified documents are threatened and could lead to various violations of privacy laws, such as HIPAA (Lord, 2015) if it is a hospital.

6.4 Qualitative analysis of subjective aspects of transparency

This chapter contains the qualitative analysis on the subjective aspects of transparency that we deem to be difficult to perform an experiment on.

6.4.1 Analysis

"The two most important transparencies are access and location transparency; their presence or absence most strongly affects the utilization of distributed resources. They are sometimes referred to together as network transparency." (Coulouris et al., 2011, p. 24)

Access Transparency

NFS

When accessing NFS, the file system must be mounted before local access can be performed and it functions as a local directory. When the file system has been mounted, there was no difference in accessing a local directory or the mounted file system. Therefore, all file operations function as they were performed on a local directory.

IPFS

IPFS was a bit more user-unfriendly as knowledge of the hash was required to mount or access the specified IPFS object. This hash is a long, unique code that is difficult for humans to remember. With this hash however, it is possible to mount the object needed and the object can be accessed in the same way as a local object. However, due to limitations in IPFS it was only possible to read from the specified object, not write to it.

Access Transparency with Security

NFS

For NFS, even with security added on top of it, there are no detriments to the access transparency. As it makes use of Access Control List (Coulouris et al., 2011, p. 480) for secure access and control over files and directories, if the user is added to the Access Control List (ACL) they will see the directories and files and will have no problem accessing the content. There will not be any checks on the users' side that they must deal with, everything is handled securely in the background.

IPFS

For IPFS, it is similar to NFS in that there will not be any checks on the user side for any content that they have access to, which gives them access to it right away. However, at the moment there is not any support for ACL or other security measures, which means that everyone can access the file. While this means the transparency is great, security suffers greatly.

Location Transparency

NFS

To mount an NFS directory, the IP or the fully qualified domain name (FQDN) is needed. When mounted, no further information is required.

IPFS

Likewise, IPFS does not require any data about the current location of the directory or file, it only requires the hash to enable access to the specified object. Regular operations can be performed on the object after it has been mounted.

Failure Transparency

NFS

NFS does not handle failures such as a server crashing. If the NFS server fails, the resources on that server will not be accessible to users.

IPFS

Depending on how the configuration of the IPFS cluster, it can handle failures such as nodes crashing. If one node has all the files that are requested, and this node crashes, the operation cannot complete. However, if multiple nodes have the files, one or more nodes can crash, and the operation are redirected to retrieve the data from the remaining nodes that still have the data.

Migration Transparency

NFS

In NFS, it can be possible for resources to be migrate if they still are accessible from the same address or IP. If these requirements are fulfilled, the client would not notice any difference when migrating resources.

IPFS

Likewise, if the hash for the specified object stays the same, the client would not notice any difference if nodes or resources are migrated. Especially since IPFS is peer-to-peer based and does not rely on one server holding all the information.

Migration Transparency with Security

NFS

As NFS makes use of ACL to control a user's access to content it means that as long as the user has access to content, the user is free to at least read and copy it. Depending on the users' rights on the content it might also be able to move the content to another directory without the user getting any prompts besides the usual that might occur (overwrite files with similar names etc.). Which leads to high migration transparency for NFS with security added.

IPFS

As for IPFS, the problem is the same as with access transparency. As long as the content is shared, the user will have full access to it and will be able to do what they want with it. This is because as of yet, there are no implementations of ACL that will restrict the content to certain nodes on the network. However, while the security is low, it means the transparency will be high, therefore, they will not have any problems moving or copying the files elsewhere.

6.4.2 Conclusion

IPFS and NFS both handle the transparencies analyzed well, even with added security, we see no added difficulties for the users. For accessing the file system, the only mandatory step was that it must be mounted, no extra steps such as an FTP-connection or equivalent operations are required. Likewise, it was not required to specify what server or node the requested files reside on.

Finally, both file systems handle faults if set up correctly. The key difference is that IPFS supports failure natively, while NFS requires third-party solutions, as the one discussed in paper by Marzullo and Schmuck (1988). Overall, IPFS may be more difficult to use as it is more user-unfriendly, due to long and unique hashes, but also requires more knowledge of how to use it. However, it is important to consider that IPFS still is relatively new and under active development and is subject to change.

In terms of the first hypothesis, the subjective aspects of transparency are as good as they are in NFS.

7 Discussion

This chapter contains a discussion on the study as a whole.

7.1 Summary

In this study we have analyzed the system attributes; scalability, performance and security of IPFS and how the file system is perceived to feel like a local file system in terms of the aspects of transparency. To analyze these system attributes, we have performed experiments with environmental simulation on scalability and performance and a qualitative analysis on security and subjective aspects of transparency.

The result of the performance experiment shows that IPFS was a lot slower than expected and does not feel like a local file system. It also does not perform well with multiple replicas of the resources. The subjective aspects of transparency, however, are well perceived in IPFS, with most issues being connected to the long and difficult to remember hashes of IPFS.

The scalability experiments produced results that were opposite to that of our hypothesis. We believed that IPFS would scale better than our results show, as it is a peer-to-peer based system. When adding replication, the overall performance in IPFS was reduced, while adding more nodes did not affect performance to the same extent, thus replication transparency was not well perceived in IPFS. However, scaling transparency was not affected by adding more replication and nodes as the underlying structure and algorithms did not change no matter how many nodes were added.

The results of the qualitative analysis of security of NFS and IPFS showed that IPFS is in its current state very weak in security. Our hypothesis that the IPFS security should be at least as good as NFS ended up becoming disproven. The goal of IPFS being a file system distributed over the web means that it requires lots of security so that systems and users do not become vulnerable to the threats that Coulouris *et al.* (2011) mentions that a distributed file system is vulnerable to. We believe, however, this is due to IPFS being in a beta state, and that better security and features will be implemented in the future, as they themselves claim it will be.

7.2 Ethical aspects in experimentation

In this study we have performed experiments on machines and qualitative analyses on different distributed file systems. Because of this, no subjects have been used in this study and such there are no issues with confidential information. The code written used to conduct the experiments has been written from scratch with the intention to only be used to be able to gather the results.

The file systems used in this study have all been open-source with sources crediting them. Nobody who have conducted the study has had any financial connection to either of the file systems which could lead the results to be biased. Further, we openly list all our results and data, and the code on how to conduct the experiments and the study. This makes it possible for anyone to conduct the same study to verify the validity of the results.

7.3 Ethical aspects in Security

When analyzing the security aspects of the file system it will show weaknesses that readers of this article might be able to use to attack vulnerable systems. However, the reverse is also true, in that it

might be able to help the developers of the systems to improve the security where it is needed and close the security holes that is brought up, if they are not already known about.

There were no actual tests performed against any vulnerabilities found, so no systems, real or not, were exposed to any attacks.

7.4 Experimental simulation results

In the performance and scalability experiments, both areas ended up with similar findings. IPFS did not perform as well as was expected and was instead a lot slower. However, we cannot verify the validity of the results because we cannot conclude if any eventual errors in our experimental simulation was the cause of the results we received.

Potential causes of the errors could be that we have multiple nodes on a single machine or that the bandwidth used to simulate a real network causes issues. Because of this, the results should not be viewed as absolute truth, and to verify our results a case study with actual IPFS nodes on individual machines would have to be conducted. If those results show the same trends, then the validity of our results can be confirmed or discarded.

The reason the validity of our results are in question is because previous research on peer-to-peer system (Bharambe, 2005) show different trends in their results. However, it is possible that it may be IPFS itself that is causing the issues. Due to these concerns, we performed a small-scale analysis, found in section 7.6.

If our results turn out to be accurate, then it may be of use for the developers of IPFS in case we managed to find a bug in how IPFS works.

7.5 Recommendations for IPFS

The results from our experiments tell us that IPFS is not well suited to be used as a distributed file system if performance is important. If fault tolerance is of high importance, then it will massively slow down the performance of IPFS. In a cluster of 100 nodes with 10 nodes replicating the data it almost doubled the execution times of common file operations on IPFS, which already was unreasonably slow.

Replication in IPFS should be limited to only a couple of nodes. This is however only tested with startup times, that is, the first time a resource is downloaded, or an operation is performed on the resource. From our results it also shows that the number of nodes in the IPFS cluster do not seem to have any major difference in performance, neither negatively or positively.

Ultimately, from our findings in experimenting with IPFS, due to it being under development and with its inadequate performance, is not well suited to be used as a distributed file system in corporations, for example. There are other distributed file systems that can be used instead with much higher performance (e.g. NFS). However, important to note. These are the conclusions we have drawn from our results. Without verifying the validity of our results, we cannot with full confidence determine that this is the case.

IPFS is not suitable for being used as a file system where security is relevant as it is. From the analysis we found that IPFS does not have all the relevant mechanisms that is necessary for it to be used as a secure file system. The lack of basic Access Control means that as soon as something is shared on the network, it is out there. IPFS also currently does not have the possibility to create private networks.

Which means that any secure information shared over IPFS can be accessed by anyone else using IPFS, even if they are not an employee of the corporation using it.

To summarize, IPFS is not suited for use cases where performance or security is of high importance. However, IPFS is well suited for use cases where availability and fault tolerance are important.

7.6 Small-scale analysis of IPFS

As stated, the performance issues of IPFS surprised us as it is a peer-to-peer file system and while we cannot with certainty say what the exact cause of the performance issues are, we conducted a small-scale test on IPFS with different settings to try to find out if the following things affected the performance heavily: number of files downloaded, the simulated bandwidth, script used to automate IPFS testing.

We performed this analysis with 10 nodes and replicating the files on 1, 2, 5 and finally 10 nodes. Each configuration was performed 100 times. This number is not enough to combat low statistical power, but it is enough to notice eventual trends. The purpose of this analysis is only to find clues as to why the performance of IPFS was lacking.

We used three different resources in our tests, with the aim to determine how it affected the settings we analyzed. These resources are:

1. Go-IPFS project unzipped (multiple directory hierarchy, 2.94MB)
2. Compressed and zipped IPFS project (Single file, 696KB)
3. Uncompressed zipped IPFS project (Single file, 2.94Mb)

Number of files downloaded

To test if the file size of the files being transferred has an impact and if a directory with multiple subdirectories is the cause of the performance issues.

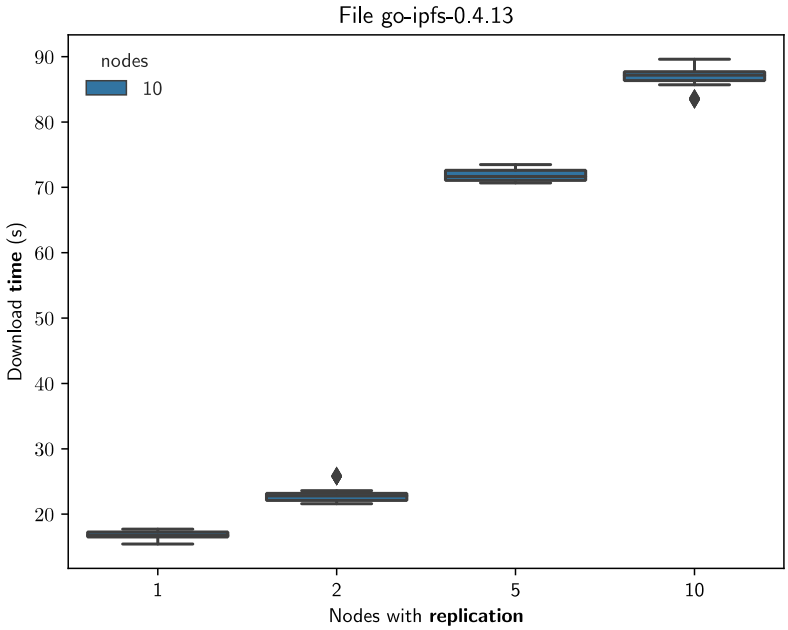

Figure 4 Download times for unzipped software project. Without bandwidth limitations.

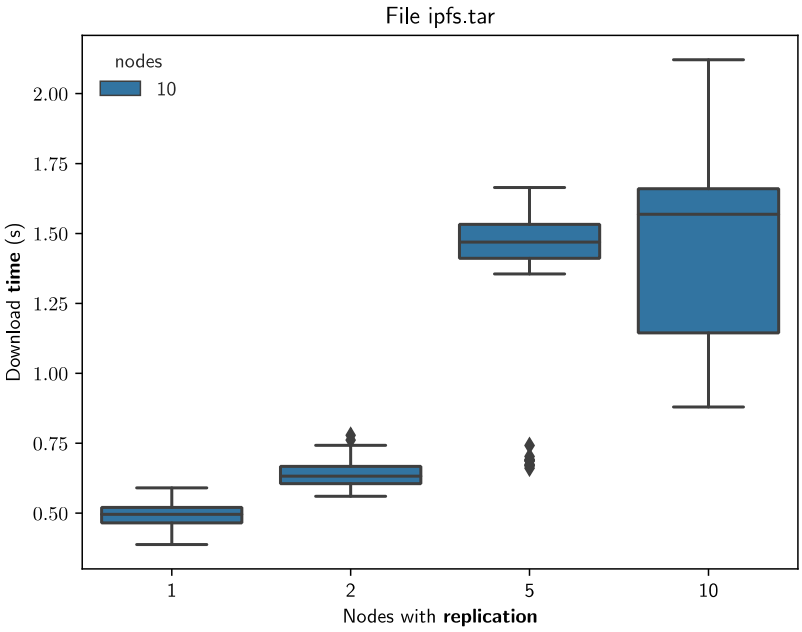

Figure 5 Download times for uncompressed software project. Without bandwidth limitations.

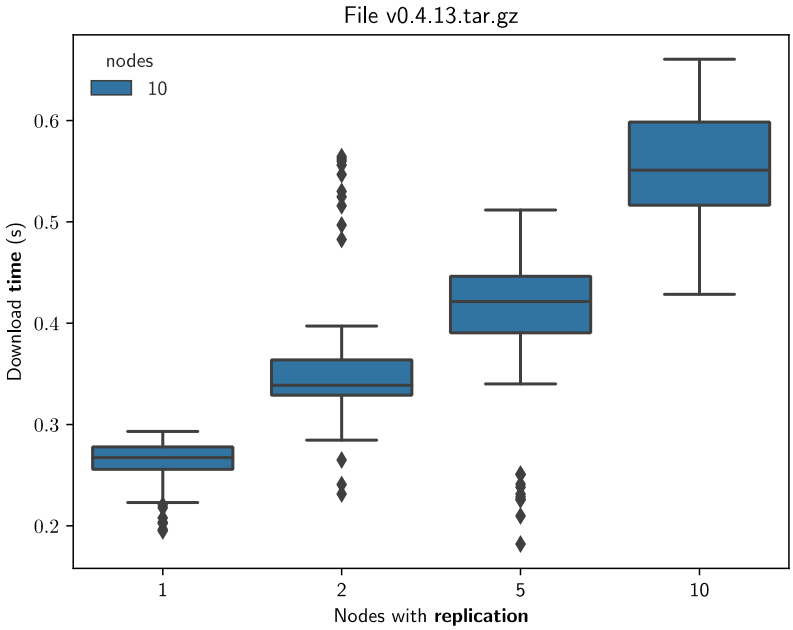

Figure 6 Download times for compressed software project. Without bandwidth limitations.

The results from this test shows that the file size does not have any major impact. However, resources with multiple subdirectories and files appeared to have a major impact in the performance. The reason for this seem to be that IPFS tries to compare and verify every downloaded file, causing high overhead. Replication does impact performance as well but does not affect the overall download times to the extent of resources with multiple files.

Comparing Figure 5 with Figure 6, we can see that the download times were improved with the smaller file size. When comparing Figure 5 with Figure 4, it is clear that resources with multiple subdirectories and files have a greater effect on performance as the download times increased significantly.

We also saw some overhead in our experimental setting when comparing results to experiments we performed on real-world setting (i.e., distributing the files between two computers on the real IPFS network). Performance in a real-world setting was better compared to the same experiment performed in our simulation. The reason for these results are uncertain since there are differences in the settings between the real-world experiment and the simulation (e.g. different operating systems and hardware). Therefore, we cannot conclude with certainty that our experimental setting had an impact on the results.

Conclusions

From this small-scale study, there are trends that nestled resources has a greater effect on performance compared to resources without nestled folders or files. Another trend appears to be that replication does not affect singular files to the same extent compared to nestled resources. It seems that the effect replication has on a singular file is added with each file in a nestled resource. This is also our hypothesis as to why IPFS performed worse than expected.

7.7 Security

IPFS was surprisingly unsecure for claiming to be a file system that should be able to be deployed over the internet. While it does work for projects where security is not important, but immunity to

takedown is, such as its use during the election in Catalonia, Spain, it does not work for much more, because everyone can access anything you share. Which means that if you want to share a document with only one other person, you must make use of external tools of encryption for said file. Which makes IPFS low in usability for a regular user.

NFS on the other hand has a lot of mechanisms for controlling access to files and directories and making sure that only an authorized entity can access the object in question. NFS however, is not suited to be deployed widely with thousands of users. It is more suited for intranet deployment where the userbase is smaller and the traffic to/from the server is not as high as it would be if it was used over the internet.

7.8 Subjective aspects of transparency

Overall, the subjective aspects are experienced well for both NFS and IPFS. The issue of IPFS is that the hashes are hard to remember which harms the location transparency. Likewise, for NFS the biggest issue is the failure transparency. Below we discuss solutions that are available for both file systems.

IPFS
There are certain implementations that help to ease the usage of IPFS and updating files or directories. The third-party implementations usually use certain blockchains (Wu, 2017) to distribute the latest hash of the specified project, but IPFS includes a tool called InterPlanetary Name Space (IPNS) that can be used to update a directory while retaining the hash. This is done by linking a hash to the address of the current node, which is also a hash, and then accessing the contents via the hash of the node.

While this can be tedious, IPNS is the official way to add dynamic content to IPFS without redistributing the hash (Benet, 2014b).

With the use of IPNS together with Domain Name System (DNS) it is possible to link the hashes to human readable names, this allows for there to be a DNS entry of “cat” to point at the hash of a picture of a cat. The picture can then be updated without the need to redistribute the hash. This improves the perceived location transparency by not having to type the whole hash.

NFS
NFS, as stated, is a client/server file system. Natively it does not support the use of multiple servers (Streicher, 2008), and such if the server crashes the file system can no longer be used. However, with the usage of third-party support, the system can be set up with multiple servers that act as backups achieving redundancy.

This way, if the primary server crashes, one of the backup servers can pick up the role of primary server and the system can continue running without the clients noticing that the original primary server had crashed, and the clients do not lose any data. This solution improves the perceived failure transparency of NFS.

7.9 An IPFS DDOS Attack

Due to the results that we received from the scalability and performance experiments, we speculate that it might be possible to use this weakness to perform a sort of DDOS attack that is similar to that of a slowloris attack (Imperva, 2018).

The attack requires that the malicious party sets as many nodes as possible that they control themselves. Then by checking which files are the most commonly accessed ones, or pre-emptively sharing files that might become popular, and spreading those files over the nodes they control it is possible to make the access to those files incredibly slow.

While this does not lead to the same damage and implications as a DDOS attack, which might bring entire servers down. It does affect the general experience of IPFS for the users by making their access to content incredibly slow, which may have the possibility for harming any service running over IPFS.

8 Future Work

8.1 Future work concerning both Scalability and Performance

As stated earlier in the report, the results of the performance and scalability experiments was very surprising and contradicting the original hypothesis. Because the experiments were performed as experimental simulations with real IPFS nodes in a local cluster with simulated network bandwidth there are possibilities that this causes issues with how IPFS works or that multiple nodes on the same machine is the cause for the results. For future work we would conduct a small-scale case study where each node is on a unique machine. The intention is to try to find out if similar trends can be seen as in our results to be able to verify the validity of our results and experimental simulation.

There is also merit to further evaluate the small-case study performed in the discussion to determine if IPFS performs worse when the shared resources contain multiple subdirectories and files.

8.2 Scalability

Another interesting future work would be to conduct a large-scale case study to confirm the result found in this study. This ties into validating the simulation environment since results should be similar if the simulation environment is valid.

8.3 Performance

Due to time limitations, interesting aspects of the performance study had to be cut out. This can be studied further as future work. One aspect is to measure how long time it takes to update files on IPFS and NFS and to see how long time it takes until the other nodes and clients using the system can notice the change. This requires knowledge in how the IPFS name service, IPNS works and there was not enough time to fully understand how it worked to conduct experiments with it. Furthermore, it can also be interesting to find out if changing larger amounts of the file has any considerate time change compared to changing small amounts.

Another interesting aspect to study is how well IPFS handles streaming. IPFS has built-in support to handle streaming content directly without the need for different applications. However, how well this behaves could be interesting to find out. Especially if there is any difference between file sizes on the files to be streamed.

8.4 Security

As the security aspect of IPFS was only a qualitative analysis of the implementations and suggested designs for the future, it would be interesting to see actual experimentation and results for how the system is affected by the implementations.

A newer qualitative analysis could be performed on IPFS later on as well, once it has left beta and been released with all the security mechanisms that it should then have.

It would also be interesting to see how well IPFS can be secured in regard to a distributed web, with things like distributed databases such as OrbitDB (2015), an IPFS based database. Further, it is also interesting to look at if it is viable to host secure web services over a decentralized distributed system.

9 References

Amber, C. (2015). Why The Internet Needs IPFS Before It's Too Late. Retrieved March 8, 2018, from https://techcrunch.com/2015/10/04/why-the-internet-needs-ipfs-before-its-too-late/

Anderson, R. (2008). Security Engineering. *John Wiley & Sons*, 313–364. https://doi.org/10.1093/epirev/mxr031

Benet, J. (2014a). go-ipfs. Retrieved April 15, 2018, from https://github.com/ipfs/go-ipfs

Benet, J. (2014b). IPFS -Content Addressed, Versioned, P2P File System. *ArXiv Preprint ArXiv:1407.3561*, (Draft 3). https://doi.org/10.1109/ICPADS.2007.4447808

Berndtsson, M., Hansson, J., Olsson, B., & Lundell, B. (2008). *Thesis Projects: A Guide for Students in Computer Science and Information Systems*. *Springer*. https://doi.org/10.1007/978-1-84800-009-4

Bharambe, A. R. (2005). Analyzing and Improving BitTorrent Performance.

Callaghan, B., Robinson, D., & Thurlow, R. (2003). *Network File System (NFS) version 4 Protocol*.

CentOS Project, T. (2017). CentOS 6.9. Retrieved May 27, 2018, from https://wiki.centos.org/Manuals/ReleaseNotes/CentOS6.9

ConsenSys. (2016). Ever Wonder How Merkle Trees Work? Retrieved May 17, 2018, from https://media.consensys.net/ever-wonder-how-merkle-trees-work-c2f8b7100ed3

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed Systems: Concepts and Design*. *Computer* (Fifth, Vol. 4). Addison Wesley. Retrieved from http://www.amazon.com/dp/0321263545

Eriksen, M. a. (2005). Trickle: A Userland Bandwidth Shaper for Unix-like Systems. *Usenix*, 61–70.

Flawed. (2018). Attacking Merkle Trees with a Second Preimage Attack. Retrieved April 25, 2018, from https://flawed.net.nz/2018/02/21/attacking-merkle-trees-with-a-second-preimage-attack/

Free Software Foundation, I. (2017). GNU Coreutils. Retrieved May 17, 2018, from https://www.gnu.org/software/coreutils/manual/html_node/index.html

Freedman, M. J., & Mazi, D. (2003). Sloppy hashing and self-organizing clusters. *Design*, *2735*, 45–55. https://doi.org/10.1007/978-3-540-45172-3_4

Gudu, D., Hardt, M., & Streit, A. (2014). Evaluating the performance and scalability of the Ceph distributed storage system. *Big Data '14 Proceedings of the IEEE International Conference on Big Data*, 177–182. https://doi.org/10.1109/BigData.2014.7004229

Guirat, F. Ben, & Filali, I. (2013). An Efficient Data Replication Approach for Structured Peer-to-Peer Systems. *Telecommunications (ICT), 2013 20th International Conference*.

Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., & West, M. J. (1988). Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*. https://doi.org/10.1145/35037.35059

Imperva. (2018). Slowloris attack. Retrieved May 31, 2018, from https://www.incapsula.com/ddos/attack-glossary/slowloris.html

Intel. (2009). Intel Xeon W3550. Retrieved April 12, 2018, from https://ark.intel.com/products/39720/Intel-Xeon-Processor-W3550-8M-Cache-3_06-GHz-4_80-GTs-Intel-QPI

Kernel.org. (2008). Ext4 Documentation. Retrieved April 24, 2018, from https://www.kernel.org/doc/Documentation/filesystems/ext4.txt

Kilburn. (2017). How the Catalan government uses IPFS to sidestep Spain's legal block. Retrieved March 9, 2018, from http://la3.org/~kilburn/blog/catalan-government-bypass-ipfs/

Kumar, M. (2018). Biggest-Ever DDoS Attack (1.35 Tbs) Hits Github Website. Retrieved March 9, 2018, from https://thehackernews.com/2018/03/biggest-ddos-attack-github.html

Lay, P. (Oracle), & Dijcks, J.-P. (Oracle). (2010). Leveraging Massively Parallel Processing in an Oracle Environment for Big Data Analytics, (November), 9. Retrieved from http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-hadoop-oracle-194542.pdf

Lord, N. (2015). What is HIPAA Compliance? Retrieved April 25, 2018, from https://digitalguardian.com/blog/what-hipaa-compliance

Marzullo, K., & Schmuck, F. (1988). Supplying high availability with a standard network file system. *Distributed Computing Systems, 1988., 8th International Conference On*, 447–453. https://doi.org/10.1109/DCS.1988.12547

Mukhopadhyay, D., Agrawal, C., Maru, D., Yedale, P., & Gadekar, P. (2014). Addressing name node scalability issue in hadoop distributed file system using cache approach. *Proceedings - 2014 13th International Conference on Information Technology, ICIT 2014*, 321–326. https://doi.org/10.1109/ICIT.2014.18

Neilsen, J. (2010). Website Response Times. Retrieved April 12, 2018, from https://www.nngroup.com/articles/website-response-times/

Nielsen, J. (1997). The Need for Speed.

OrbitDB. (2015). OrbitDB. Retrieved April 23, 2018, from https://github.com/orbitdb/orbit-db

Python Software Foundation, I. (2018). Python 3 - Random.

Red Hat, I. (2017). Chapter 6. The ext4 file system. Retrieved May 26, 2018, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/storage_administration_guide/ch-ext4

Regeringskansliet. (2016). Sverige helt uppkopplat 2025 – en bredbandsstrategi.

Rodrigues, B., Bocek, T., & Stiller, B. (2017). Enabling a Cooperative , Multi-domain DDoS Defense by a Blockchain Signaling System. *Semantic Scholar*.

Streicher, M. (2008). Scale your file system with Parallel NFS Read and write hundreds of gigabytes per second, *2008*, 1–8.

TechTerms. (2012). Bandwidth. Retrieved April 24, 2018, from https://techterms.com/definition/bandwidth

Treat, T. (2017). Comcast. Retrieved April 12, 2018, from https://github.com/tylertreat/Comcast

Trifa, Z., & Khemakhem, M. (2014). Sybil nodes as a mitigation strategy against sybil attack. *Procedia Computer Science*, *32*, 1135–1140. https://doi.org/10.1016/j.procs.2014.05.544

Verhoef, R., & Grinfelds, U. (2011). Transfer.sh. Retrieved April 25, 2018, from https://transfer.sh/

Wang, F., Nelson, M., Oral, S., Atchley, S., Weil, S., Settlemyer, B. W., … Hill, J. (2013). Performance and Scalability Evaluation of the Ceph Parallel File System. *PDSW '13 Proceedings of the 8th Parallel Data*

Storage Workshop, 14–19. https://doi.org/10.1145/2538542.2538562

Weil, S. a. (2007). Ceph: Reliable, Scalable, and High-Performance Distributed Storage. *Annals of Physics*, *129*, 239. Retrieved from http://ceph.newdream.net/weil-thesis.pdf

Whyrusleeping. (2017). Content Encryption. Retrieved March 1, 2018, from https://github.com/ipfs/notes/issues/270

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering. Experimentation in Software Engineering* (Vol. 9783642290). https://doi.org/10.1007/978-3-642-29044-2

Wu, H. (2017). IPFS-Ethereum-Storage. Retrieved April 10, 2018, from https://github.com/howardwu/IPFS-Ethereum-Storage

Xiong, L., Goryczka, S., & Sunderam, V. (2011). Adaptive, secure, and scalable distributed data outsourcing: a vision paper. *Proceedings of the 2011 Workshop on Dynamic Distributed Data-Intensive Applications, Programming Abstractions, and Systems*, 1–6. https://doi.org/10.1145/1996010.1996012

Appendix A - Scalability experiment

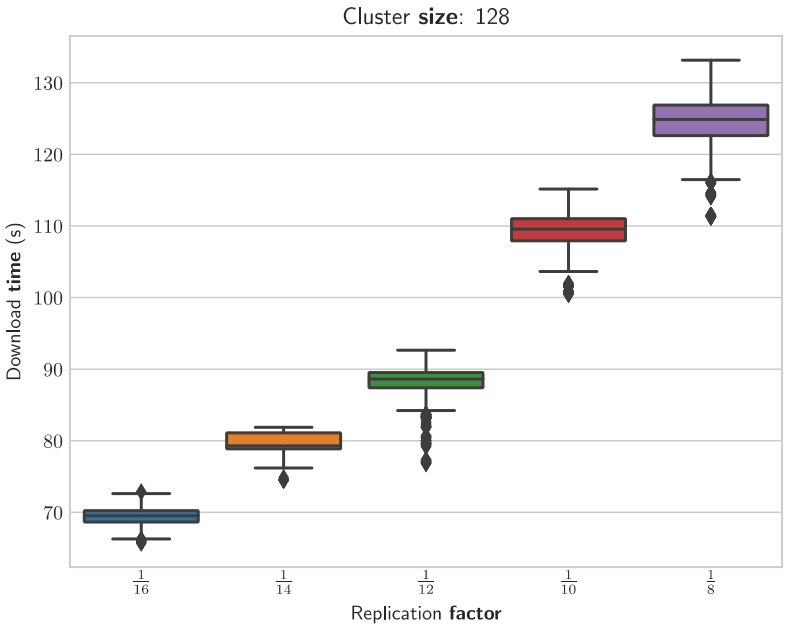

Figure 7 Average download times with varying replication factor for a cluster size of 128.

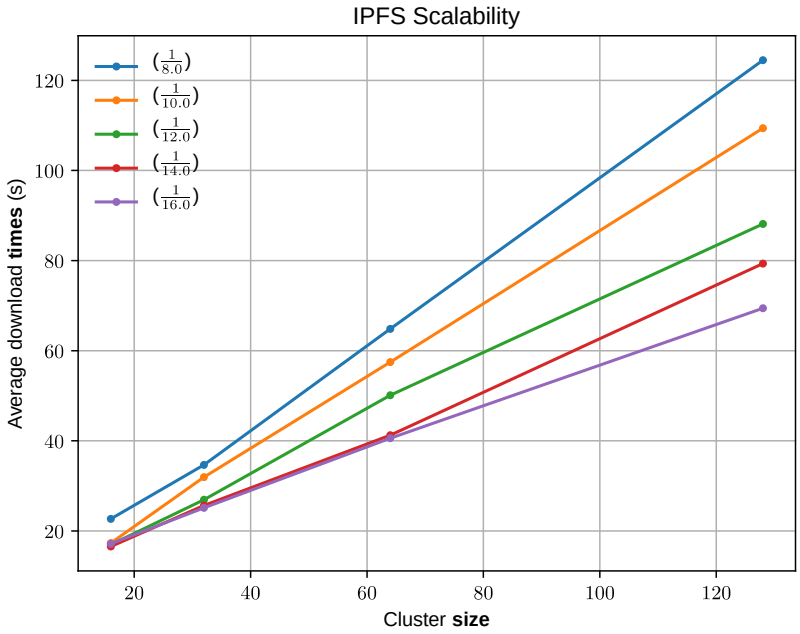

Figure 8 Average download times with varying replication factor and cluster size

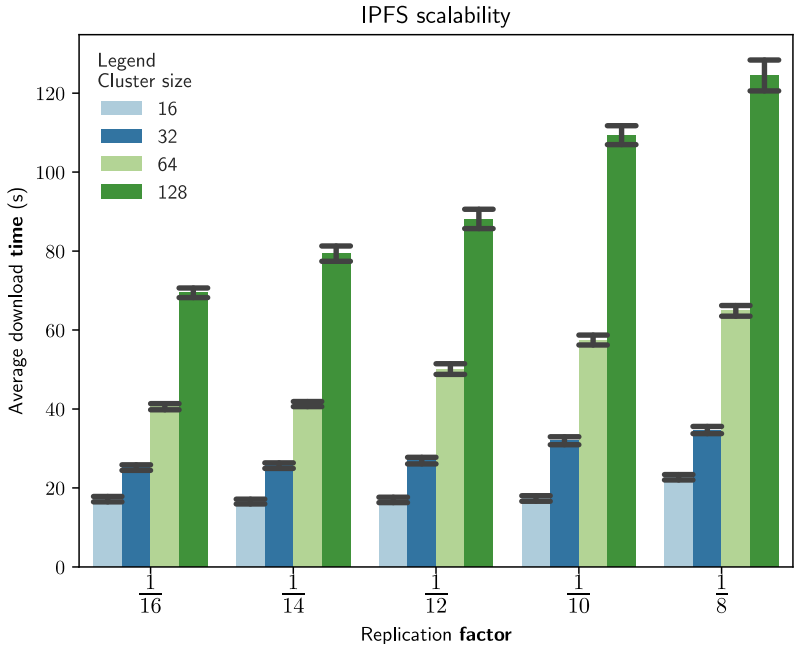

Figure 9 Average download times with varying replication sizes and cluster sizes

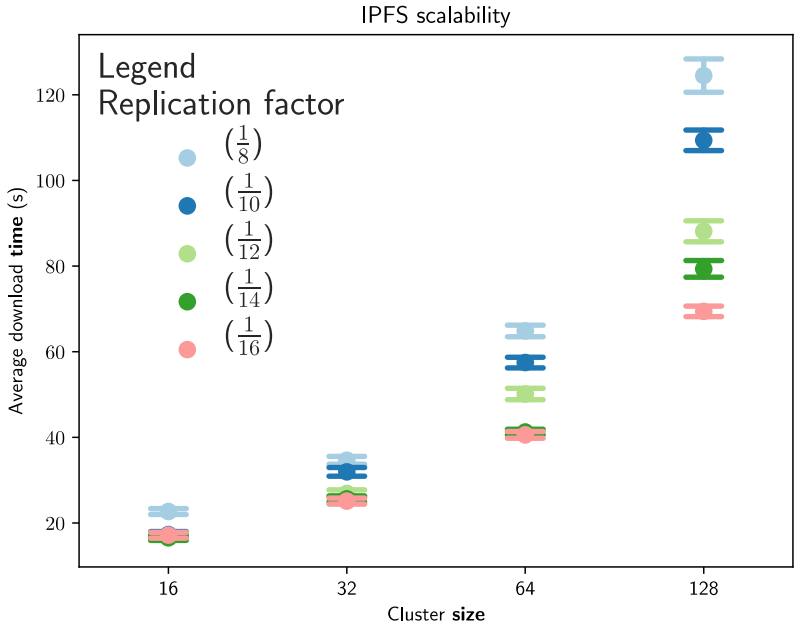

Figure 10 Average download times for each replication factor.

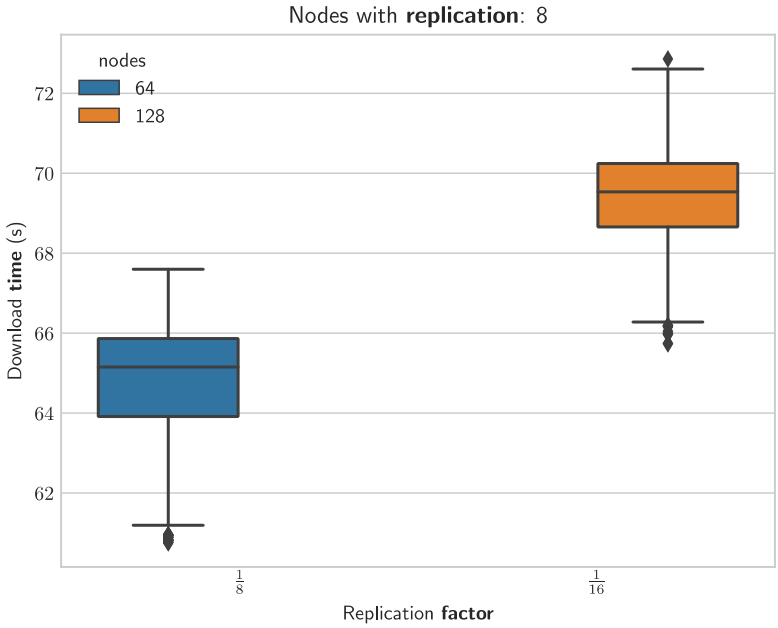

Figure 11 Average download times with replication factors $\frac{1}{8}$ and $\frac{1}{16}$ with a cluster size of 64 and 128, respectively

Appendix B - Make Directory

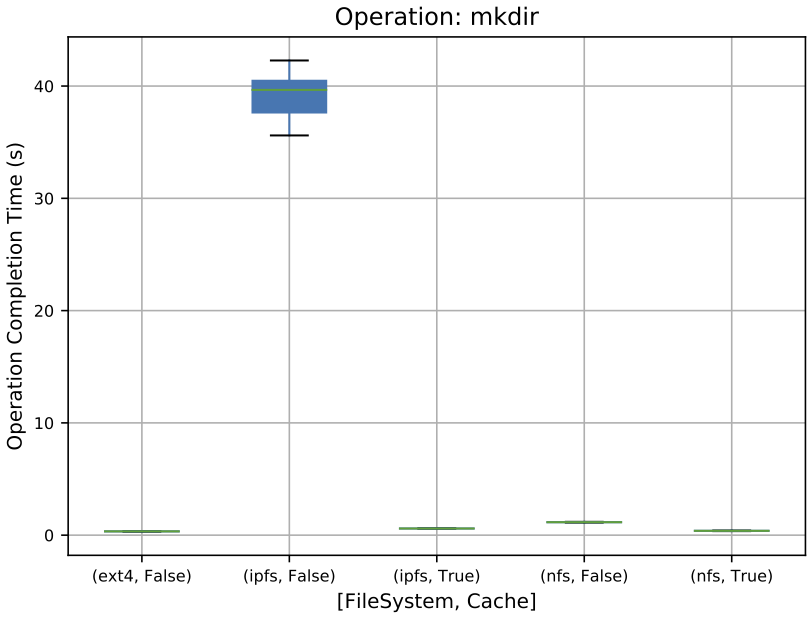

Figure 12 Completion time in seconds for Make directory with file system and cache

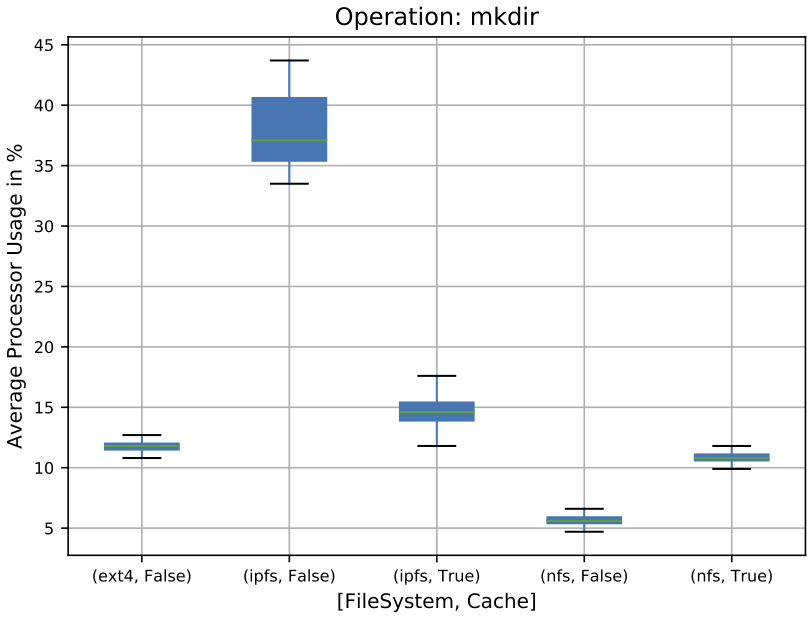

Figure 13 Processor usage in percent for Make directory with file system and cache

Appendix C - Copy

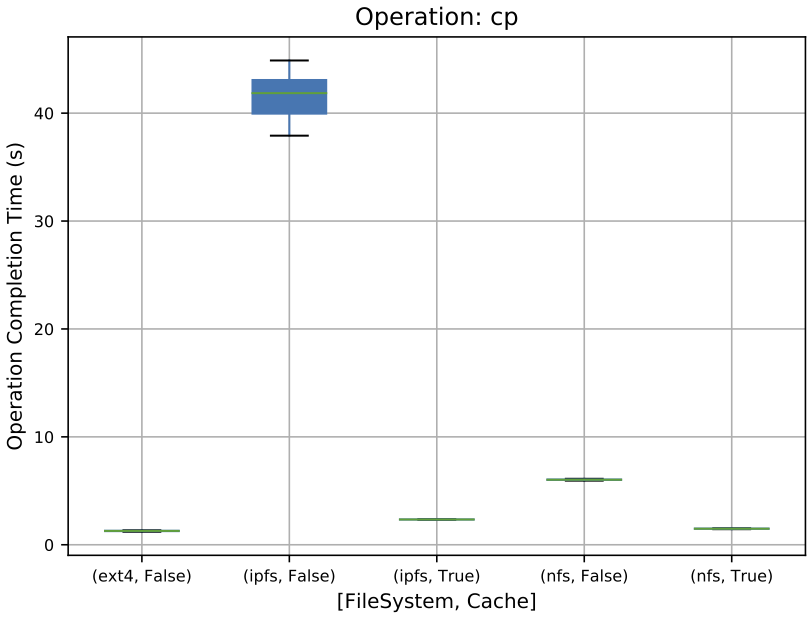

Figure 14 Completion time in seconds for Copy with file systems and cache. This graph is the same as Figure 2.

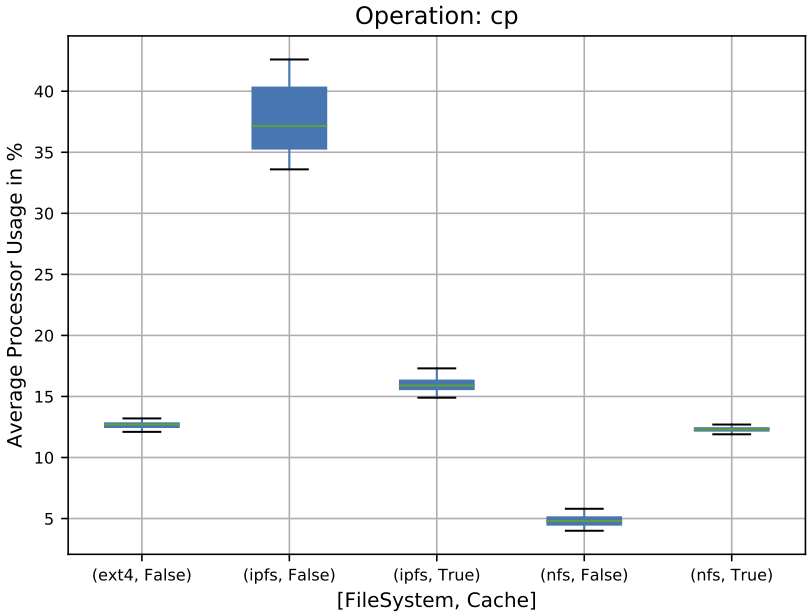

Figure 15 Processor usage in percent for Copy operation with file system and cache

Appendix D - Scan

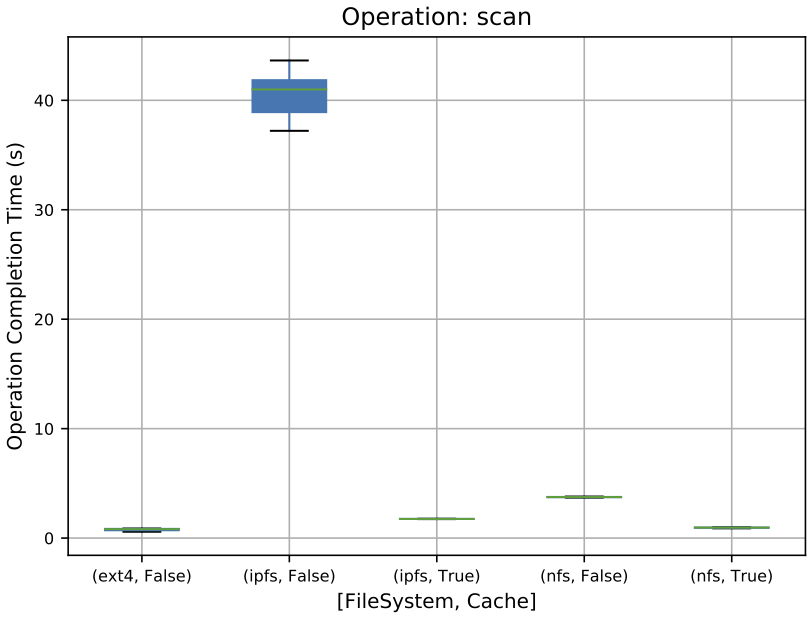

Figure 16 Completion time in seconds for Scan operation with file system and cache

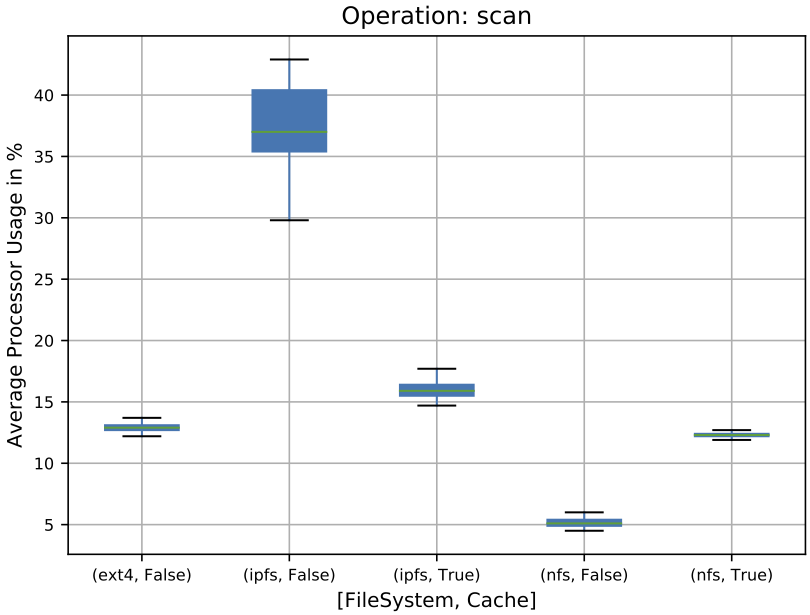

Figure 17 Processor usage in percent for Scan operation with file system and cache

Appendix E - Stat

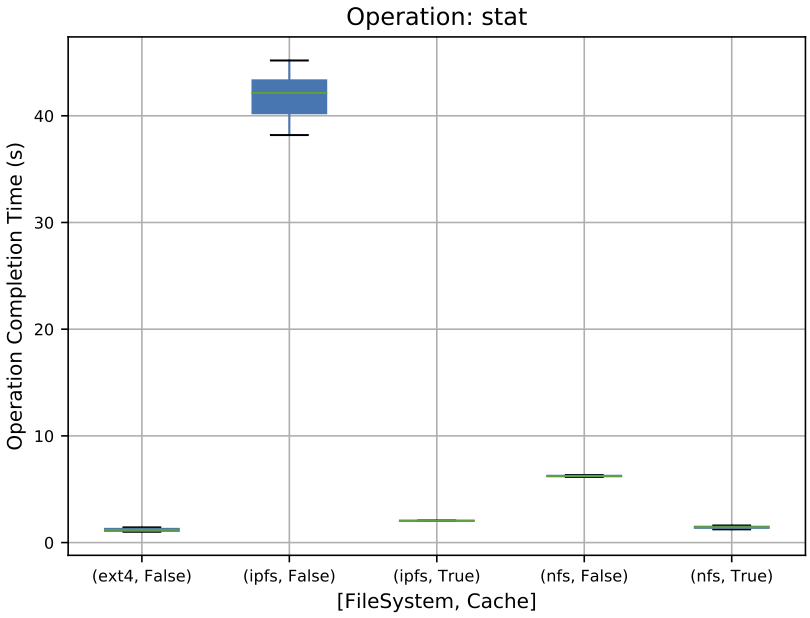

Figure 18 Completion time in seconds for Stat operation with file system and cache

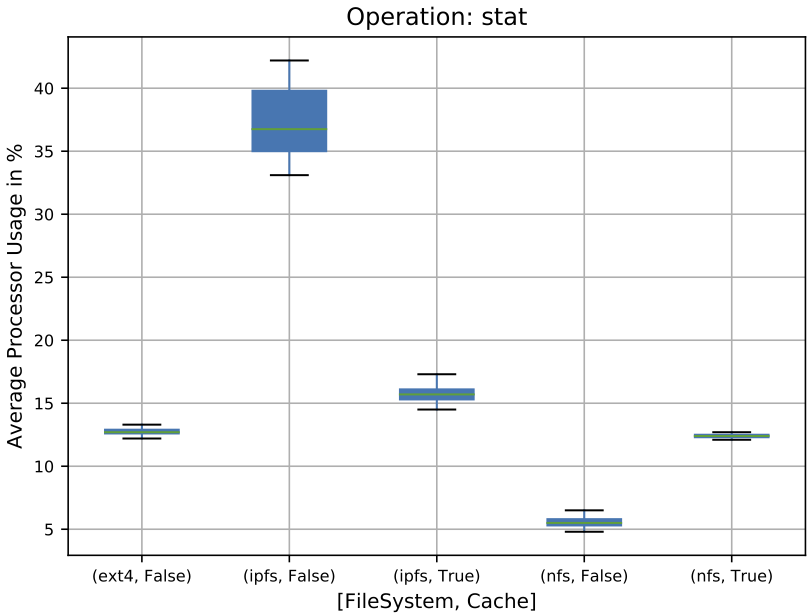

Figure 19 Processor usage in percent for Stat operation with file system and cache

Appendix F - Bandwidth

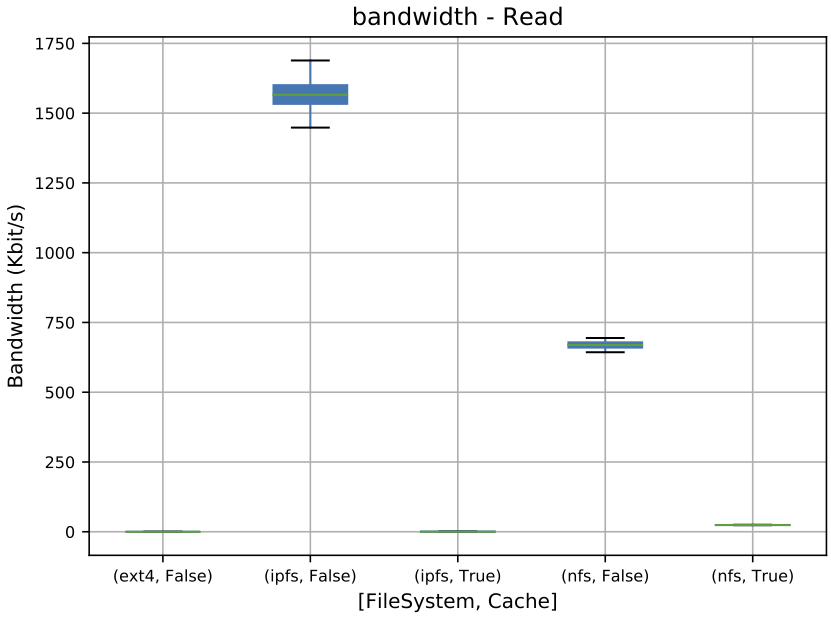

Figure 20 Bandwidth read usage in Kbit/s with file system and cache for all four operations

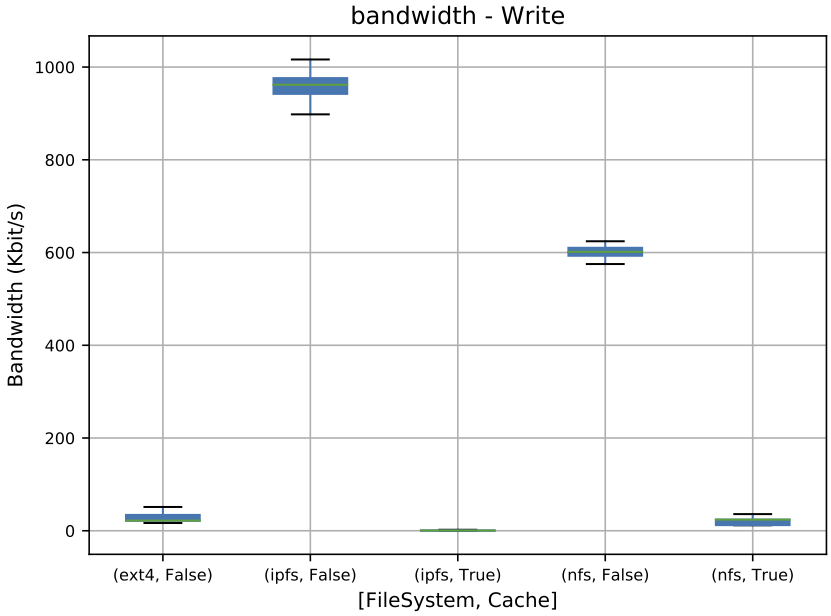

Figure 21 Bandwidth write usage in Kbit/s with file system and cache for all four operations

Appendix G - IPFS Make Directory

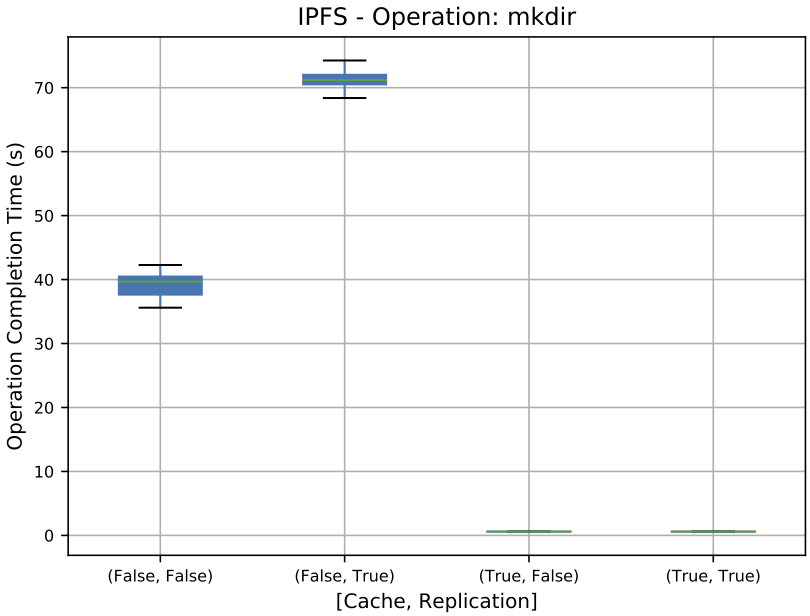

Figure 22 Completion time in seconds for Make directory in IPFS with cache and replication

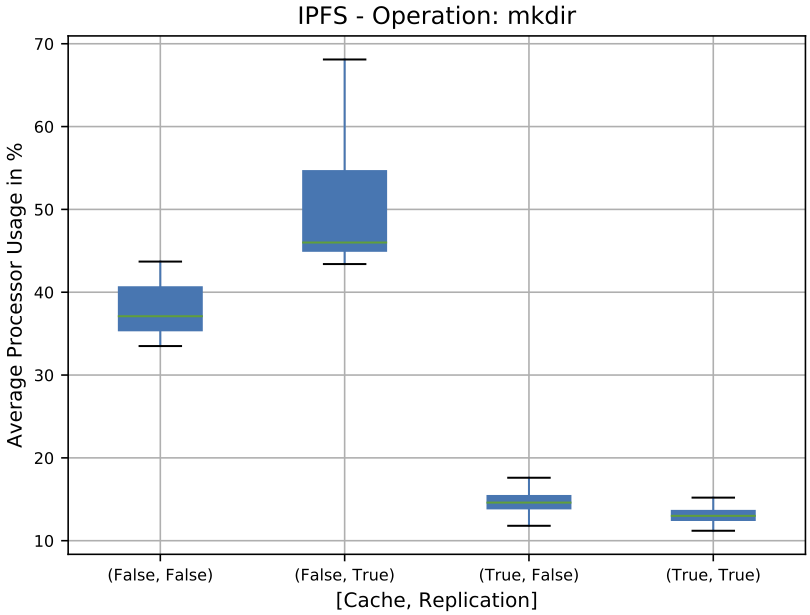

Figure 23 Processor usage in percent for Make directory operation in IPFS with cache and replication

Appendix H - IPFS Copy

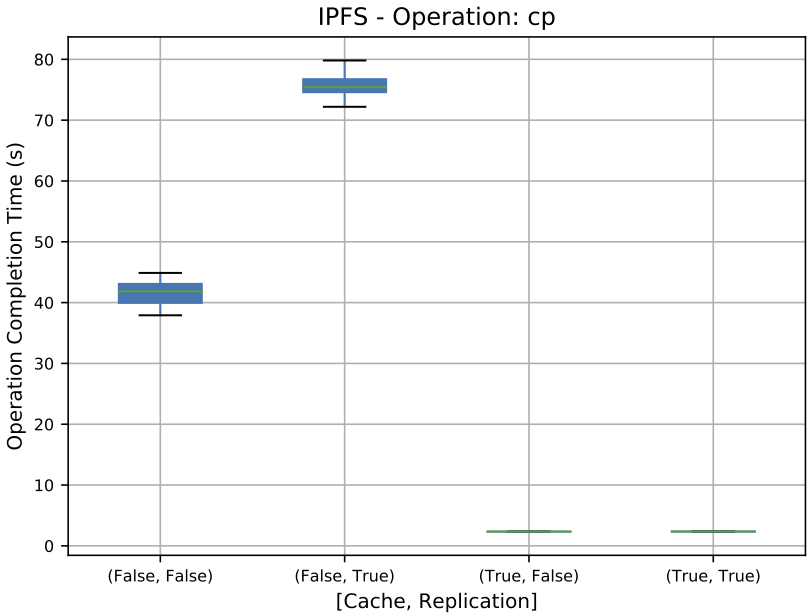

Figure 24 Completion time in seconds for Copy operation in IPFS with cache and replication. This graph is the same as Figure 3

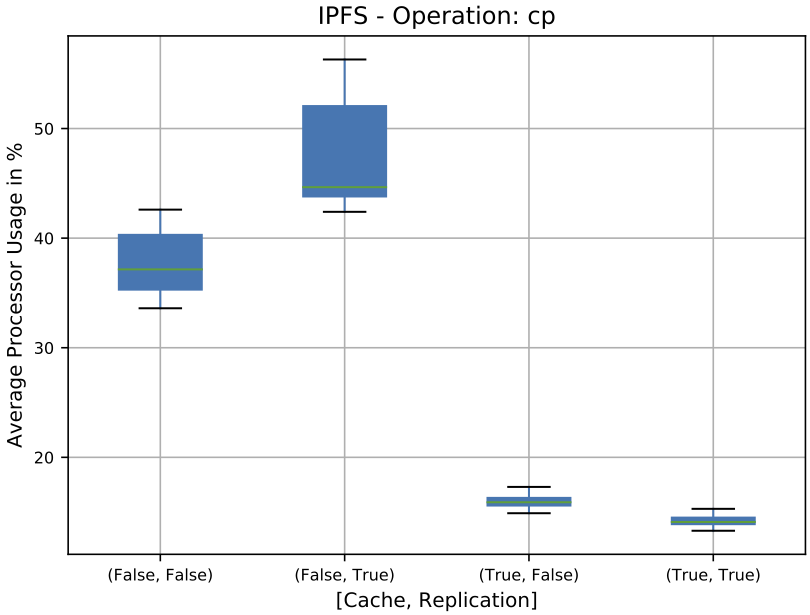

Figure 25 Processor usage in percent for Copy operation in IPFS with cache and replication

Appendix I - IPFS Scan

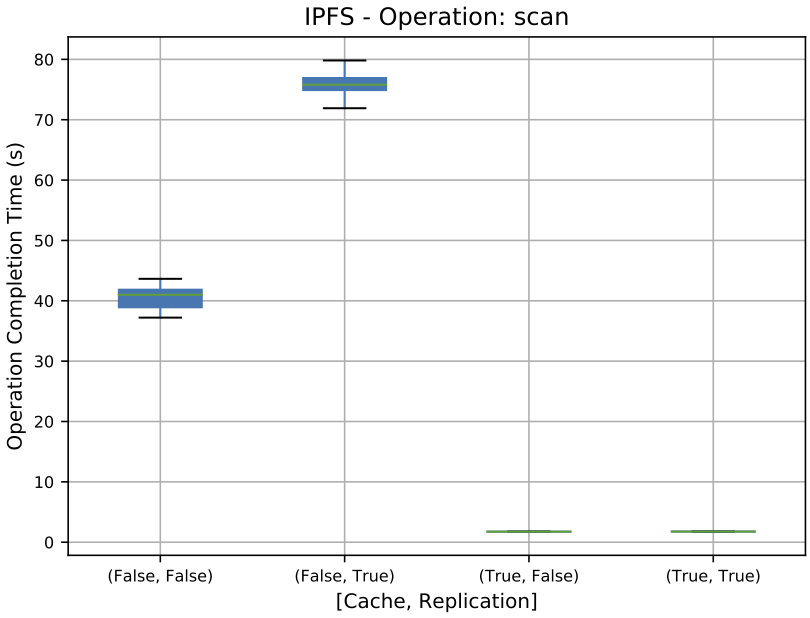

Figure 26 Completion time in seconds for Scan operation in IPFS with cache and replication

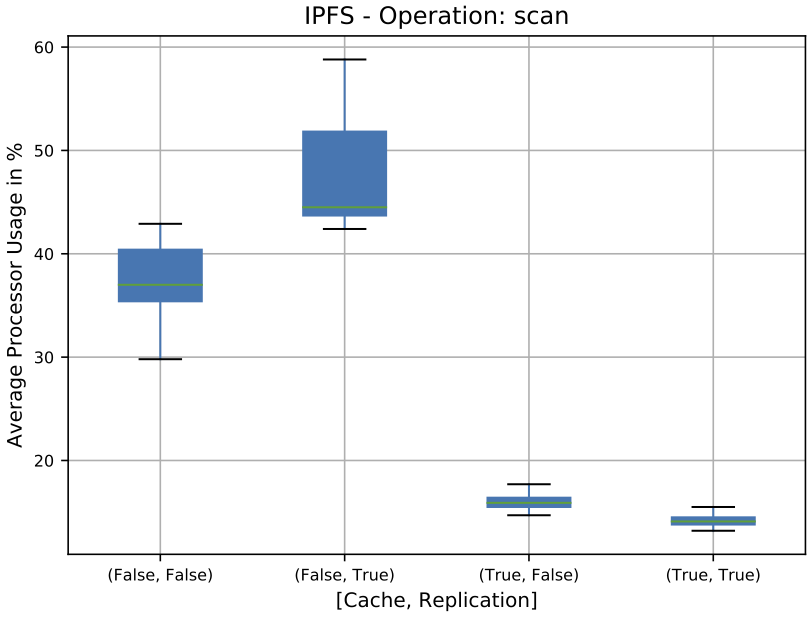

Figure 27 Processor usage in percent for Scan operation in IPFS with cache and replication

Appendix J - IPFS Stat

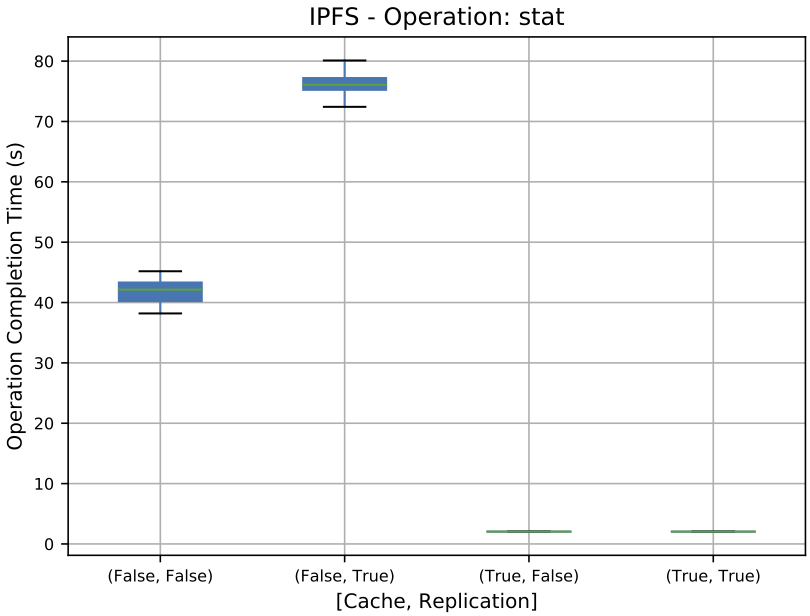

Figure 28 Completion time in seconds for Stat operation in IPFS with cache and replication

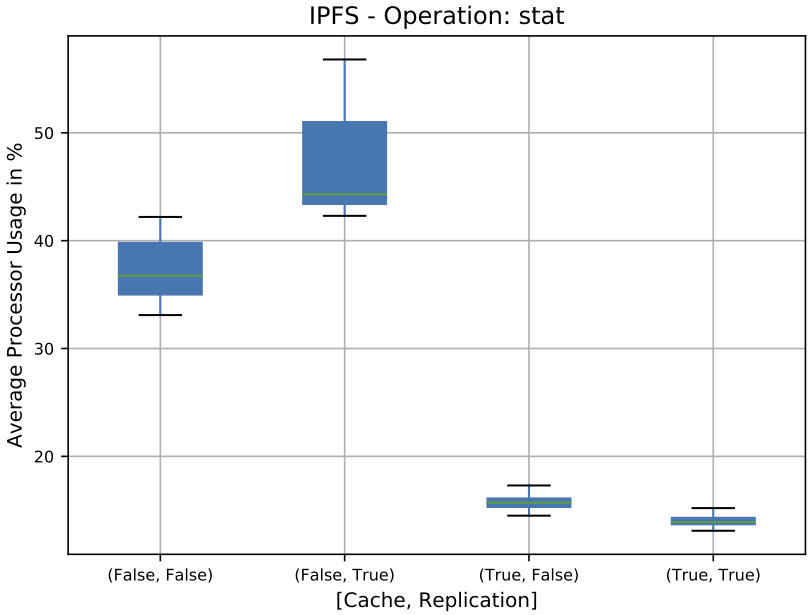

Figure 29 Processor usage in percent for Stat operation in IPFS with cache and replication

Appendix K - IPFS Bandwidth

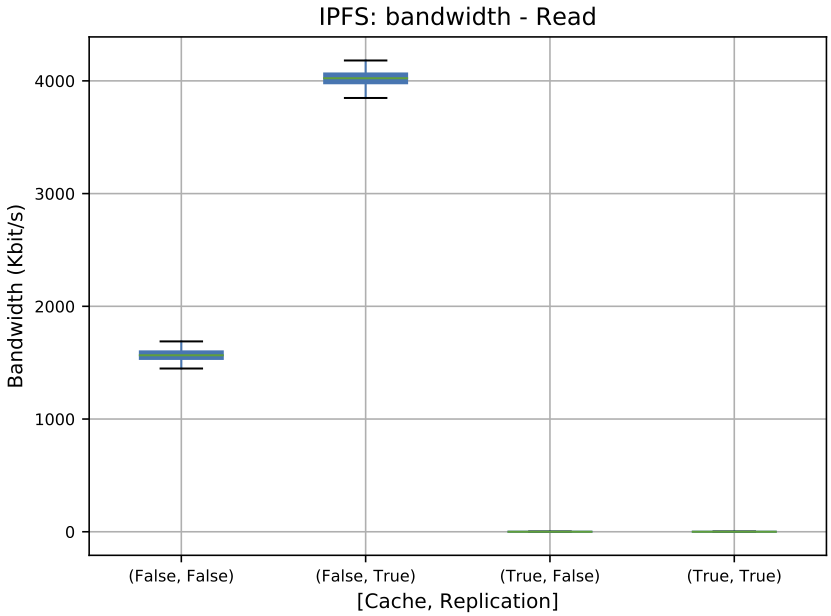

Figure 30 Bandwidth read usage in Kbit/s for IPFS with cache and replication for all four operations

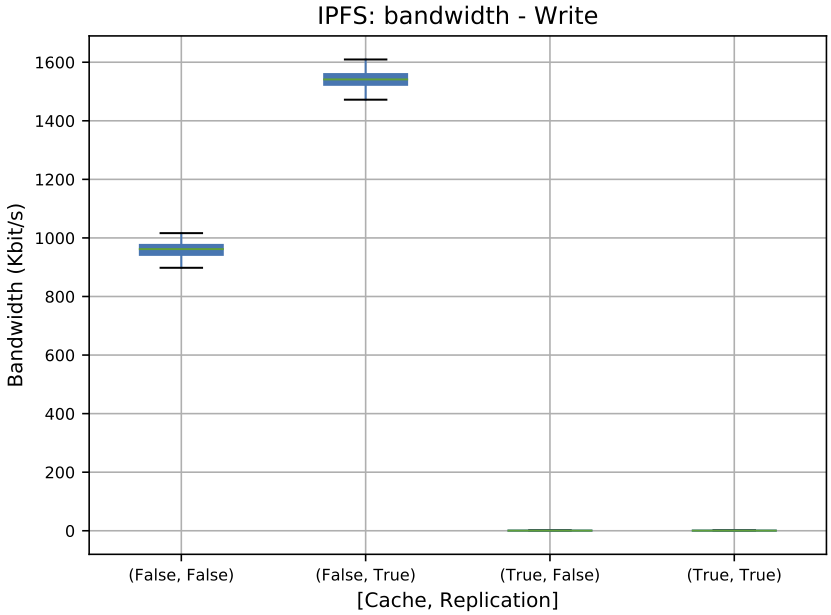

Figure 31 Bandwidth write usage in Kbit/s for IPFS with cache and replication for all four operations

Appendix L - Performance data analysis

								Time			
								Std Dev	Median	L. Quartile	U. Quartile
Filesystem	ext4	Operation	cp	Cache	false	Replication	false	0.05	1.28	1.26	1.30
			mkdir	Cache	false	Replication	false	0.02	0.34	0.33	0.35
			scan	Cache	false	Replication	false	0.08	0.81	0.71	0.84
			stat	Cache	false	Replication	false	0.10	1.19	1.10	1.29
	ipfs	Operation	cp	Cache	false	Replication	false	1.72	41.85	39.95	43.07
							true	1.58	75.42	74.59	76.74
					true	Replication	false	0.06	2.34	2.34	2.35
							true	0.07	2.35	2.35	2.36
			mkdir	Cache	false	Replication	false	1.56	39.66	37.64	40.47
							true	1.24	71.19	70.52	72.02
					true	Replication	false	0.01	0.59	0.59	0.60
							true	0.01	0.59	0.58	0.59
			scan	Cache	false	Replication	false	1.66	40.99	38.92	41.84
							true	1.54	75.80	74.91	76.91
					true	Replication	false	0.05	1.75	1.75	1.76
							true	0.05	1.76	1.76	1.77
			stat	Cache	false	Replication	false	1.66	42.14	40.23	43.32
							true	1.43	76.08	75.24	77.20
					true	Replication	false	0.05	2.05	2.04	2.05
							true	0.06	2.04	2.04	2.05
	nfs	Operation	cp	Cache	false	Replication	false	0.09	6.03	6.01	6.05
					true	Replication	false	0.04	1.49	1.48	1.50
			mkdir	Cache	false	Replication	false	0.03	1.15	1.15	1.16
					true	Replication	false	0.01	0.39	0.38	0.39
			scan	Cache	false	Replication	false	0.02	3.75	3.73	3.76
					true	Replication	false	0.04	0.96	0.93	0.96
			stat	Cache	false	Replication	false	0.04	6.24	6.22	6.26
					true	Replication	false	0.08	1.44	1.38	1.49

Figure 32 Performance data

Appendix M - Performance processor usage

								Processorusage			
								Std Dev	Median	L. Quartile	U. Quartile
Filesystem	ext4	Operation	cp	Cache	false	Replication	false	0.22	12.70	12.50	12.80
			mkdir	Cache	false	Replication	false	0.48	11.80	11.50	12.00
			scan	Cache	false	Replication	false	0.32	12.90	12.70	13.10
			stat	Cache	false	Replication	false	0.33	12.70	12.60	12.90
	ipfs	Operation	cp	Cache	false	Replication	false	2.63	37.15	35.30	40.30
							true	4.39	44.65	43.80	52.10
					true	Replication	false	0.54	15.90	15.60	16.30
							true	0.47	14.10	13.90	14.50
			mkdir	Cache	false	Replication	false	2.66	37.10	35.40	40.60
							true	5.03	46.00	45.00	54.65
					true	Replication	false	1.17	14.60	13.90	15.40
							true	0.98	13.00	12.50	13.60
			scan	Cache	false	Replication	false	2.69	37.00	35.40	40.40
							true	4.29	44.50	43.70	51.85
					true	Replication	false	0.58	15.90	15.50	16.40
							true	0.56	14.10	13.80	14.50
			stat	Cache	false	Replication	false	2.58	36.75	35.00	39.80
							true	4.34	44.30	43.40	51.00
					true	Replication	false	0.58	15.70	15.30	16.10
							true	0.50	13.90	13.70	14.30
	nfs	Operation	cp	Cache	false	Replication	false	0.39	4.80	4.50	5.10
					true	Replication	false	0.21	12.30	12.20	12.40
			mkdir	Cache	false	Replication	false	0.39	5.60	5.40	5.90
					true	Replication	false	0.52	10.80	10.60	11.10
			scan	Cache	false	Replication	false	0.29	5.10	4.90	5.40
					true	Replication	false	0.23	12.30	12.20	12.40
			stat	Cache	false	Replication	false	0.30	5.50	5.30	5.80
					true	Replication	false	0.20	12.40	12.30	12.50

Figure 33 Performance processor usage data

Appendix N - Performance bandwidth usage

								Read				Write			
								Std Dev	Median	L. Quartile	U. Quartile	Std Dev	Median	L. Quartile	U. Quartile
Filesystem	ext4	Operation	bandwidth	Cache	false	Replication	false	25.31	0.00	0.00	0.00	29.50	22.00	21.43	34.34
	ipfs	Operation	bandwidth	Cache	false	Replication	false	52.93	1565.62	1533.18	1600.36	22.48	961.89	942.12	976.23
							true	71.42	4023.28	3978.47	4065.75	29.56	1541.34	1522.82	1559.55
					true	Replication	false	0.95	0.14	0.00	0.80	1.04	0.51	0.00	0.87
							true	0.83	0.18	0.00	0.77	0.50	0.50	0.00	0.87
	nfs	Operation	bandwidth	Cache	false	Replication	false	12.00	669.28	659.83	678.71	10.54	601.56	593.03	610.25
					true	Replication	false	13.99	24.10	23.89	24.37	7.41	23.90	12.11	24.19

Figure 34 Performance bandwidth usage data

Appendix O - Scalability: nodes over replication factor

				Time				
				Mean	Std Dev	Median	L. Quartile	U. Quartile
Nodes	16.0	Replication Factor	8.0	22.69	0.68	22.82	22.40	23.16
			10.0	17.31	0.70	17.31	16.80	17.91
			12.0	16.95	0.68	17.03	16.55	17.48
			14.0	16.55	0.60	16.56	16.16	17.01
			16.0	17.16	0.69	17.17	16.68	17.68
	32.0	Replication Factor	8.0	34.66	0.91	34.66	34.18	35.10
			10.0	31.95	1.01	31.92	31.43	32.50
			12.0	26.92	0.83	27.00	26.67	27.45
			14.0	25.65	0.70	25.69	25.19	26.19
			16.0	25.14	0.70	25.24	24.74	25.65
	64.0	Replication Factor	8.0	64.85	1.34	65.15	63.91	65.87
			10.0	57.47	1.25	57.54	56.68	58.44
			12.0	50.12	1.34	50.39	49.60	50.93
			14.0	41.25	0.66	41.27	40.91	41.65
			16.0	40.59	0.78	40.57	40.10	41.15
	128.0	Replication Factor	8.0	124.48	3.90	124.88	122.61	126.89
			10.0	109.37	2.40	109.55	107.91	111.03
			12.0	88.14	2.45	88.60	87.39	89.53
			14.0	79.35	1.96	79.29	78.86	81.11
			16.0	69.42	1.22	69.53	68.66	70.25

Figure 35 Number of nodes with varying replication factors.

Appendix P - Replication factor over number of nodes

				Time				
				Mean	Std Dev	Median	L. Quartile	U. Quartile
Replication Factor	8.0	Nodes	16.0	22.69	0.68	22.82	22.40	23.16
			32.0	34.66	0.91	34.66	34.18	35.10
			64.0	64.85	1.34	65.15	63.91	65.87
			128.0	124.48	3.90	124.88	122.61	126.89
	10.0	Nodes	16.0	17.31	0.70	17.31	16.80	17.91
			32.0	31.95	1.01	31.92	31.43	32.50
			64.0	57.47	1.25	57.54	56.68	58.44
			128.0	109.37	2.40	109.55	107.91	111.03
	12.0	Nodes	16.0	16.95	0.68	17.03	16.55	17.48
			32.0	26.92	0.83	27.00	26.67	27.45
			64.0	50.12	1.34	50.39	49.60	50.93
			128.0	88.14	2.45	88.60	87.39	89.53
	14.0	Nodes	16.0	16.55	0.60	16.56	16.16	17.01
			32.0	25.65	0.70	25.69	25.19	26.19
			64.0	41.25	0.66	41.27	40.91	41.65
			128.0	79.35	1.96	79.29	78.86	81.11
	16.0	Nodes	16.0	17.16	0.69	17.17	16.68	17.68
			32.0	25.14	0.70	25.24	24.74	25.65
			64.0	40.59	0.78	40.57	40.10	41.15
			128.0	69.42	1.22	69.53	68.66	70.25

Figure 36 Replication factor and cluster size

Appendix Q - Small-scale analysis

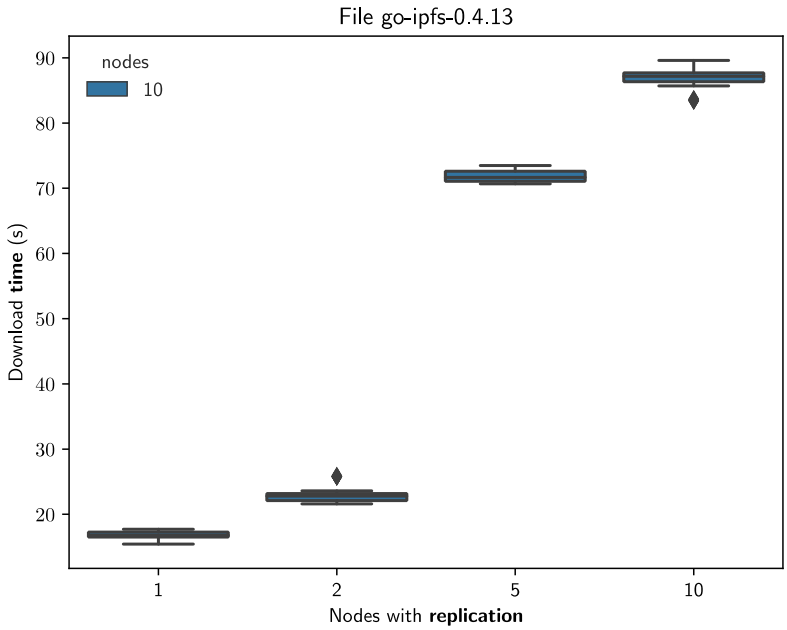

Figure 37 Download times for non-zipped project. Without bandwidth limitations

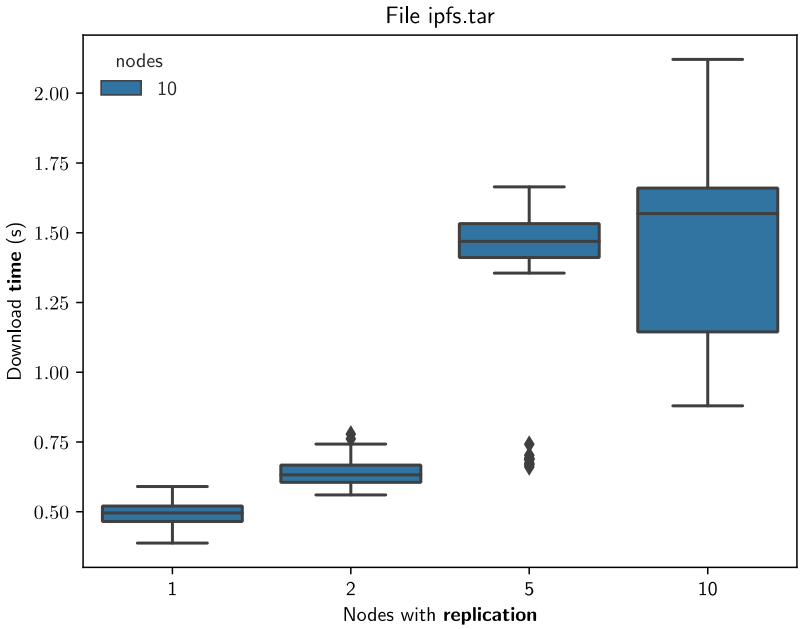

Figure 38 Download times for uncompressed zipped project. Without bandwidth limitations

Appendix R - deploy.sh

```
#! /usr/bin/env bash
DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

STATS_FILE="stats.csv"
head -n 1 "$DIR/template" >"$DIR/$STATS_FILE"

DEV=lo
DEV1=enp1s0
# KBITSPEED=12800 # 100Mbit in Kbit
KBITSPEED=12500
CLIENTS=1
REPLICATION=6
# NODES=32
CLUSTER_NODES=(16 32 64)
MYIP="$(ip add | grep -Eo 'inet (addr:)?([0-9]*\.){3}[0-9]*' | grep -Eo
'([0-9]*\.){3}[0-9]*' | grep -v '127.0.0.*')"
tc qdisc del dev "$DEV" root netem
tc qdisc del dev "$DEV1" root netem
umount /ipfs
umount /ipns
rm -rf "$DIR/deploy/ipfs*"
APIPORT=5001
APILIST=()
pids=()
rm -rf "$DIR/deploy/ipfs0"
mkdir -p "$DIR/deploy/ipfs0"
export IPFS_PATH="$DIR/deploy/ipfs0"
ipfs init -e --profile test &>/dev/null &
pids+=($!)
ipfs bootstrap rm all &>/dev/null &
unset IPFS_PATH
wait "${pids[@]}"

export IPFS_PATH="$DIR/deploy/ipfs0"
ipfs config Addresses.Gateway /ip4/0.0.0.0/tcp/8080
ipfs config Datastore.StorageMax 0GB
ipfs config --json Datastore.StorageGCWatermark 0
ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin "[\"*\"]"
ipfs    config    --json    API.HTTPHeaders.Access-Control-Allow-Methods
"[\"PUT\", \"POST\", \"GET\"]"
ipfs  config  --json  API.HTTPHeaders.Access-Control-Allow-Credentials
"[\"true\"]"
ipfs config Datastore.GCPeriod 0h
ipfs config Addresses.API /ip4/0.0.0.0/tcp/"$APIPORT"
APILIST+=("$APIPORT")
sed -i 's/127.0.0.1/0.0.0.0/g' "$IPFS_PATH/config"
trickle -s -u "$KBITSPEED" -d "$KBITSPEED" ipfs daemon --enable-gc=true
>"$IPFS_PATH/daemon.stdout" 2>"$IPFS_PATH/daemon.stderr" &
echo $! >"$IPFS_PATH/daemon.pid"
echo "Starting gateway node"
unset IPFS_PATH

STARTED=0
while ((STARTED < CLIENTS)); do
```

```
    STARTED=0
    for requsts in "${APILIST[@]}"; do
        if ! curl -fs "http://localhost:$requsts"; then
            ((STARTED++))
        fi
    done
    sleep 1
done
echo "Done starting daemons"
NODE_0_ADDR="$(curl -s http://localhost:5001/api/v0/id?format=\<id\> |
jq '.Addresses[0]' | cut -d "\"" -f 2)"

API="http://localhost:$APIPORT/api/v0"
curl -sSn "$API/bootstrap/add?arg=${NODE_0_ADDR}" &>/dev/null
curl -sSn "$API/swarm/connect?arg=${NODE_0_ADDR}" &>/dev/null
echo "Done bootstrapping $((CLIENTS)) clients.."
NODE_0_ADDR="$(curl -s http://localhost:5001/api/v0/id?format=\<id\> |
jq '.Addresses[0]' | cut -d "\"" -f 2 | sed "s/127.0.0.1/${MYIP}/")"
IFS=' ' read -r -a array <<<"$@"
for NODES in "${CLUSTER_NODES[@]}"; do
    export IPFS_PATH="$DIR/deploy/ipfs0"
    readarray -t client_nodes < <(ipfs swarm peers)
    for client_node in "${client_nodes[@]}"; do
        ipfs swarm disconnect "$client_node" &> /dev/null
    done
    unset IPFS_PATH
    rm -rf "$DIR/clients_*"
    for cluster in "${array[@]}"; do
        ssh -n -f root@"$cluster" bash -c "'(cd /root/ipfs-scalability;
nohup bash /root/ipfs-scalability/deploy_cluster.sh $NODE_0_ADDR $NODES
> /root/ipfs-scalability/daemon.out 2>&1 & echo $! > /root/ipfs-
scalability/daemon.pid) &'"
    done
    sleep 2
    for cluster in "${array[@]}"; do
        while true; do
            rm -rf "$DIR/clients_$cluster.txt"
            if scp root@"$cluster:/root/ipfs-scalability/clients.txt"
"$DIR/clients_$cluster.txt" &>/dev/null; then
                break
            fi
            sleep 3
        done
    done

    IPFS_FILE="$(find $DIR/files/* -maxdepth 0 -type d -exec basename {}
\;)"
    IPFS_FILE_SIZE="$(du -sh "$DIR/files/go-ipfs-0.4.13" | awk '{ print
$1 }')"
    ITERATIONS=1000
    pids=()
    echo "python3 "$DIR/node_download.py" $((${#array[@]} * NODES))
$ITERATIONS $IPFS_FILE $IPFS_FILE_SIZE $REPLICATION"
    python3 "$DIR/node_download.py" "$((${#array[@]} * NODES))"
"$ITERATIONS" "$IPFS_FILE" "$IPFS_FILE_SIZE" "$REPLICATION"
done
```

Appendix S - deploy_cluster.sh

```
#! /usr/bin/env bash
DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

DEV=lo
DEV1=enp1s0
# KBITSPEED=12800 # 100Mbit in Kbit
KBITSPEED=125000
eval NODE="$2"
MYIP="$(ip add | grep -Eo 'inet (addr:)?([0-9]*\.){3}[0-9]*' | grep -Eo
'([0-9]*\.){3}[0-9]*' | grep -v '127.0.0.*')"
rm -rf "$DIR/clients.txt"
printf "Running... %s \n" "$(date)" >"$DIR/running.txt"

printf "client\n" >"$DIR/client.txt"

kill -9 "$(cat "$DIR/deploy.pid")" &> /dev/null
echo $$ >"$DIR/deploy.pid"

tc qdisc del dev "$DEV" root netem
tc qdisc del dev "$DEV1" root netem
rm -rf "$DIR/ipfs_*"
APILIST=()
pids=()
for ((i = 0; i < NODE; i++)); do
    echo "$DIR/ipfs_$i"
    rm -rf "$DIR/ipfs_$i"
    mkdir -p "$DIR/ipfs_$i"
    export IPFS_PATH="$DIR/ipfs_$i"
    ipfs init -e --profile test &>/dev/null &
    pids+=($!)
    ipfs bootstrap rm all &>/dev/null &
    unset IPFS_PATH
done
wait "${pids[@]}"
for ((i = 0; i < NODE; i++)); do
    export IPFS_PATH="$DIR/ipfs_$i"
    ipfs   config   --json   API.HTTPHeaders.Access-Control-Allow-Origin
"[\"*\"]"
    ipfs config --json API.HTTPHeaders.Access-Control-Allow-Credentials
"[\"true\"]"
    ipfs   config   --json   API.HTTPHeaders.Access-Control-Allow-Methods
"[\"PUT\", \"POST\", \"GET\"]"
    ipfs config Addresses.API /ip4/0.0.0.0/tcp/0
    trickle   -s   -u   "$KBITSPEED"   -d   "$KBITSPEED"   ipfs   daemon
>"$IPFS_PATH/daemon.stdout" 2>"$IPFS_PATH/daemon.stderr" &
    echo $! >"$IPFS_PATH/daemon.pid"
    echo "Trying to find API port for node $i"
    while true; do
        if grep "API" "$IPFS_PATH/daemon.stdout" &> /dev/null; then
            node_port="$(grep  "API"  "$IPFS_PATH/daemon.stdout"  |  awk
'{split($5,a,"/"); print a[5] }')"
            echo "Found API port for node $i..."
            APILIST+=( "$node_port" )
            break
```

```
        fi
        sleep 1
    done
    printf "http://%s:%s\n" "$MYIP" "$node_port" >>"$DIR/client.txt"
    echo "Starting node $i"
    unset IPFS_PATH
done

echo "Done starting daemons"
NODE_0_ADDR=$1
echo "${NODE_0_ADDR}"

for api_port in "${APILIST[@]}"; do
    API="http://localhost:$api_port/api/v0"
    curl -sSn "$API/bootstrap/add?arg=${NODE_0_ADDR}" &>/dev/null
    curl -sSn "$API/swarm/connect?arg=${NODE_0_ADDR}" &>/dev/null
done
echo "Done bootstrapping $((NODE)) nodes.."

mv "$DIR/client.txt" "$DIR/clients.txt"

cat
```

Appendix T - node_download.py

```
#!/usr/bin/env python3
""" Script to run simulation tests on IPFS using ipfsapi and python """

import glob
import multiprocessing as mp
import os
import random
import subprocess
import sys
import time
from contextlib import contextmanager
from itertools import product
from urllib.parse import urlparse

import ipfsapi
# import numpy as np
import pandas as pd
from filelock import FileLock, Timeout

dir_path = os.path.dirname(os.path.realpath(__file__))
file_out = os.path.join(dir_path, "stats.csv")
file_out_lock = os.path.join(dir_path, "stats.csv.lock")
lock = FileLock(file_out_lock, timeout=2)
# nodes = sys.argv[3:]


@contextmanager
def suppress_stdout():
    """ Suppresses output """
    with open(os.devnull, "w") as devnull:
        old_stdout = sys.stdout
        sys.stdout = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout


def get_clients(replication):
    """ Extract all clients from the .csv files """
    all_files = glob.glob(os.path.join(dir_path, "clients_*.txt"))
    concatenated_df = pd.DataFrame()
    list = []
    for file_ in all_files:
        df = pd.read_csv(file_, index_col=None, header=0)
        list.append(df)
    # df_from_each_file = (pd.read_csv(f) for f in all_files)
    concatenated_df = pd.concat(list)
    # concatenated_df = pd.concat(df_from_each_file, ignore_index=True)
    selected_nodes = []
    size = concatenated_df.shape[0]
    dist = random.sample(range(0, size), int(size / int(replication)))
    for i in dist:
        selected_nodes.extend(concatenated_df.iloc[i])
```

```
    return selected_nodes


def subprocess_cmd(command):
    """ Runs command as subprocess """
    # start_time = time.time()
    process    =    subprocess.Popen(command,    stdout=subprocess.PIPE,
shell=True)
    process.communicate()[0].strip()
    # time_string = str(time.time() - start_time) + "," + nodes + '\n'
    # file.write(time_string)


def upload_files(node, q):
    """ Uploads all files to given  """
    node_url = urlparse(node)  # Parses the given URL
    print("{}:{}".format(node_url.hostname, node_url.port))
    ipfs_node = ipfsapi.connect(node_url.hostname, node_url.port)
    res    =    ipfs_node.add(dir_path    +    '/files/go-ipfs-0.4.13',
recursive=True)
    q.put(res[-1]['Hash'])


def download_files(node, iterations, gateway, ipfs_hash, file_size, file,
                   nr_nodes, replication):
    """ Download the given file from IPFS """
    print("Downloading {} times from node {} ".format(iterations, node))
    move_path = os.path.join(dir_path, "newDir_{}".format(node))
    subprocess_cmd("rm -rf {}".format(move_path))
    for _ in range(0, int(iterations)):
        subprocess_cmd("mkdir -p {}".format(move_path))
        start_time = time.time()
        subprocess_cmd("ipfs get {} -o {}".format(ipfs_hash, move_path))
        time_string = str(
            time.time() - start_time
        ) + "," + file_size + "," + file + "," + nr_nodes + "," +
"1/{}".format(
            replication) + '\n'
        lock.acquire()
        try:
            open(file_out, "a").write(time_string)
        finally:
            lock.release()
        subprocess_cmd("rm -rf {}".format(move_path))
        gateway.repo_gc()


def    scalability_test(nr_nodes,    iterations,    file,    file_size,
replication):
    """ Main method for scalability test """
    nodes = get_clients(replication)
    processes = []
    queue = mp.Queue()
    for node in nodes:
        p = mp.Process(target=upload_files, args=(node, queue))
        processes.append(p)
        p.start()
    for process in processes:
```

```
        process.join()
    print("Done adding files to nodes")
    gateway = ipfsapi.connect('127.0.0.1', 5001)
    os.environ["IPFS_PATH"] = os.path.join(dir_path, "deploy", "ipfs0")
    IPFS_HASH = queue.get(0)
    print(IPFS_HASH)
    nodes = 10
    node_download = []
    node_iterations = int(iterations) / int(nodes)
    for node in range(0, int(nodes)):
        p = mp.Process(
            target=download_files,
            args=(node, node_iterations, gateway, IPFS_HASH, file_size,
file,
                  nr_nodes, replication))
        node_download.append(p)
        p.start()
    for node in node_download:
        node.join()


if __name__ == '__main__':
    scalability_test(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4],
sys.argv[5])
```

Appendix U - Performance_experiment.py

```
""" Script to be used when running experiments on ext4, NFS and IPFS """
import os
import random
import string
import subprocess
import sys
import time

import psutil

dir_path = os.path.dirname(os.path.realpath(__file__))
originalEditFile = os.path.join(dir_path, "supaFile.txt")
FNULL = open(os.devnull, 'w')
file = open(os.path.join(dir_path, "performance_stats.csv"), "a")


def subprocess_editFile(command):
    """ Run a subprocess and return output """
    print(command)
    process    =    subprocess.Popen(command,    stdout=subprocess.PIPE,
shell=True, env=os.environ)
    out = process.communicate()[0].strip()
    return out

def execute_bw_stats(filesystem):
    if filesystem == "ext4":
        bw_stats = subprocess_editFile("awk '/sda2/ {print $6 / 2,$10 /
2}' /proc/diskstats | tail -n1")
        bw_stats = bw_stats.decode('ascii').split(" ")
    if filesystem == "nfs":
        bw_stats  =  subprocess_editFile("awk  '/enp1s0/  {print  $2  /
1024,$10 / 1024}' /proc/net/dev")
        bw_stats = bw_stats.decode('ascii').split(" ")
    if filesystem == "ipfs":
        bw_stats         =         subprocess_editFile("curl         -s
http://localhost:5001/api/v0/stats/bw? | jq '.TotalIn,.TotalOut'")
        bw_stats = bw_stats.decode('ascii').split("\n")
        bw_stats[0] = int(bw_stats[0]) / 1024
        bw_stats[1] = int(bw_stats[1]) / 1024

    return bw_stats

def write_to_stats(operation, time, filesystem, processor_usage, cache,
replication, bwread, bwwrite):
    stats_string = operation + "," + time + "," + filesystem + "," +
processor_usage + "," + cache + "," + replication + "," + str(bwread) +
"," + str(bwwrite) + '\n'
    file.write(stats_string)
    file.flush()

def subprocess_cmd(command, operation, fileSystem, cache, replication):
    """ Run subprocess and time execution time """
    print(command)
    p = None
```

```
    if fileSystem is "nfs":
        pid = subprocess.check_output(["pgrep", "-f", "nfsclient"])
        p = psutil.Process(pid)
    elif fileSystem is "ipfs":
        pid = subprocess.check_output(["pgrep", "-f", "ipfs"])
        p = psutil.Process(pid)
    else:
        p = psutil
    p.cpu_percent(interval=1)
    start_time = time.time()
    process    =    subprocess.Popen(command,    stdout=subprocess.PIPE,
shell=True, env=os.environ)
    process.communicate()[0].strip()
    completionTime = str(time.time() - start_time)
    processorUsage        =        str(psutil.cpu_percent(interval=None,
percpu=False))

    timeString = operation + "," + completionTime + "," + fileSystem +
"," + processorUsage + "," + cache + "," + replication + 'null' + 'null'
+ '\n'
    if operation != "rmrf":
        write_to_stats(operation,      completionTime,      fileSystem,
processorUsage, cache, replication, 'null', 'null')


def ipfsClearCache(cache):
    if cache == 'false':
        subprocess_editFile('ipfs repo gc')


def runExperiment(parameters):
    with open('performance_stats.csv') as f:
        first_line = f.readline()
    if                                                                  not
(first_line.__contains__("Operation,Time,FileSystem,ProcessorUsage,Cach
e,Replication,Read,Write")
            ):

file.write("Operation,Time,FileSystem,ProcessorUsage,Cache,Replication,
Read,Write\n")


    splitArgs = str(parameters).split(", ")
    fileSystem = removeExtraLines(splitArgs[1])
    filePath_From = removeExtraLines(splitArgs[2])
    iterations = int(removeExtraLines(splitArgs[3]))
    cache = removeExtraLines(splitArgs[4])
    replication = removeExtraLines(splitArgs[5])
    working_dir = "/root/go/src/github.com/ipfs/go-ipfs"
    if (fileSystem == "ext4" or fileSystem == "nfs"):

        for _ in range(0, iterations):
            subprocess_cmd('rm -rf %s' % working_dir, 'rmrf',
                           fileSystem, cache, replication)  # RemoveDir

            start_time = time.time()
            before_out = execute_bw_stats(fileSystem)
            before_read = before_out[0]
```

```
            before_write = before_out[1]
            subprocess_cmd(
                'cd %s ; find . -type d -exec mkdir -p %s/{} \;'
                % (filePath_From, working_dir), "mkdir", fileSystem,
cache, replication)  # MakeDir

            subprocess_cmd(
                'cd %s ; find . -type f -exec cp {} %s/{} \;'
                % (filePath_From, working_dir), "cp", fileSystem, cache,
replication)  # Copy all

            subprocess_cmd(
                'cd %s ; find . -type f -exec stat {} \;' % filePath_From,
                "stat", fileSystem, cache, replication)  # Status

            subprocess_cmd(
                'cd %s ; find . -type f -exec cat {} \;' % filePath_From,
                "scan", fileSystem, cache, replication)  # Read all

            complete_time = str(time.time() - start_time)
            after_out = execute_bw_stats(fileSystem)
            after_read = after_out[0]
            after_write = after_out[1]
            read_bw = (float(after_read) - float(before_read)) /
float(complete_time)
            write_bw = (float(after_write) - float(before_write)) /
float(complete_time)
            print(read_bw, write_bw)
            write_to_stats('bandwidth', 'null', fileSystem, 'null',
cache, replication, read_bw, write_bw)
        print("Finished")
    elif fileSystem == "ipfs":
        os.environ["IPFS_PATH"] = os.path.join("/root/ipfs-scalability",
"deploy", "ipfs0")
        for _ in range(0, iterations):
            subprocess_cmd('rm -rf %s' % working_dir, 'rmrf',
                           fileSystem, cache, replication)  # RemoveDir
            ipfsClearCache(cache)

            start_time = time.time()
            before_out = execute_bw_stats(fileSystem)
            before_read = before_out[0]
            before_write = before_out[1]
            subprocess_cmd(
                'cd %s ; find . -type d -exec mkdir -p %s/{} \;'
                % (filePath_From, working_dir), "mkdir", fileSystem,
cache, replication)  # MakeDir
            ipfsClearCache(cache)

            subprocess_cmd(
                'cd %s ; find . -type f -exec cp {} %s/{} \;'
                % (filePath_From, working_dir), "cp", fileSystem, cache,
replication)  # Copy all
            ipfsClearCache(cache)

            subprocess_cmd(
                'cd %s ; find . -type f -exec stat {} \;' % filePath_From,
                "stat", fileSystem, cache, replication)  # Status
```

```
            ipfsClearCache(cache)

            subprocess_cmd(
                'cd %s ; find . -type f -exec cat {} \;' % filePath_From,
                "scan", fileSystem, cache, replication)  # Read all
            ipfsClearCache(cache)

            complete_time = str(time.time() - start_time)
            after_out = execute_bw_stats(fileSystem)
            after_read = after_out[0]
            after_write = after_out[1]
            read_bw  =  (float(after_read)  -  float(before_read))  /
float(complete_time)
            write_bw  =  (float(after_write)  -  float(before_write))  /
float(complete_time)
            print(read_bw, write_bw)
            write_to_stats('bandwidth',  'null',  fileSystem,  'null',
cache, replication, read_bw, write_bw)
        print("ipfs_finished")
    else:
        print(
            "Incorrect file system. Must be ext4, ipfs or nfs. Only
lowercase!"
            + " You wrote: " + fileSystem)


def createFile(length):
    """ Create file of specified length """
    for i in range(0, length):
        writeFile.write(random.choice(string.ascii_letters))


def removeExtraLines(parameter):
    for char in "'" "[" "]" ",":
        parameter = parameter.replace(char, "")
    return parameter


if __name__ == '__main__':
    if (len(sys.argv) >= 6 and len(sys.argv) < 7):
        start_time = time.time()
        runExperiment(sys.argv)
        print(str(time.time() - start_time))
    else:
        print(
            "Wrong parameters. File System, /Path of original dir, Number
of iterations, Cache, Replication. Seperate with commas"
        )
```