DISTRIBUTED FILE SYSTEM BY ADVANTAGING ONLINE WEB SERVICES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Glenn Olsson

June 2022

COMMITTEE MEMBERSHIP

TITLE:	Distributed file system by advantaging online web services
AUTHOR:	Glenn Olsson
DATE SUBMITTED:	June 2022
COMMITTEE CHAIR:	Zachary Peterson, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Aaron Keen, Ph.D. Professor of Computer Science

ABSTRACT

Distributed file system by advantaging online web services

Glenn Olsson

Today there are free online services that can be used to store files of arbitrary types and sizes, such as Google Drive. These services are often limited by a certain total storage size. My goal is to create a filesystem that similarly can store arbitrary amount and types of data but without any real limit. This is to be achieved by taking advantage of online webpages such as Twitter where text and files can be posted on free accounts with no visible limit. The goal is to have a filesystem that behaves like any other but where the actual data is stored for free on unsuspecting websites.

ACKNOWLEDGMENTS

Thanks to:

- Andrew Guenther, for uploading this template
- I would like to thank xxxx for having yyyy.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Project Overview

This project intends to create a filesystem called *Fejk File System* (FFS) which takes advantage of online web services such as Twitter. The idea is to save the files by posting or sending an encrypted version of the file as posts or private messages on these services. The intention is not to create a revolutionary fast and usable filesystem but instead to explore how well it is possible to utilize the storage that Twitter and similar services provides for free as a filesystem. The performance and limits will however be analyzed and compared to existing alternatives such as Google Drive to compare the benefits of this free storage compared to a professional system that might cost money.

1.2 Background

Year after year, people increase their total data storage used for obvious reasons. Cameras get better leading to images and videos taking more space, and with storage being cheap and easily usable, files are not needed to be deleted meaning that the data usage accumulates This means that users will require more and more storage throughout their lifetime, and even potentially beyond their lifetime if dependants want to keep these files. System storage in our hardware devices often increases with new product cycles. Today you can keep hundreds of gigabytes in your pocket without spending a big fortune. Along with increasing device storage is cloud storage increasing. For instance Apple's service iCloud allows users to store up to 2TB of data in the cloud for a few bucks per month . Even though the cost per month is not a lot, after many months this cost accumulates and you as a user get more and more dependent on this storage, especially as you don't want to spend time looking

through all your data and maybe remove some to save space . With increased pricing or necessary space upgrade, the cost will be even higher.

Social media platforms such as Twitter, Flickr, and Facebook have many millions of daily users that post anything from texts to images for their cats or funny videos. According to Henna Kermani at Twitter, they processed about 200GB of image data every second in 2016[1]. A single user posting a few images per day does not significantly change the amount of data processed or saved at all for these tech giants - a few gigabytes here or there will probably go unnoticed. . The difference between the photos posted on Twitter compared to the ones stored on cloud services such as iCloud is that the images on Twitter are stored for free for the users, indefinitely. While iCloud and similar services often have a free-tier of storage, Twitter does not have an upper limit of how many images or tweets one can make

1.3 Problem

Is it possible to create a free distributed filesystem that takes advantage of online services to store the data? Can this filesystem be competitive compared to commercial available solutions that cost money?

1.4 Purpose

The purpose of this paper is to explore the possibility to store data for free on online services, and to compare the performance of such a system to a professional filesystem service that costs money. The interesting aspect of this is that services such as social medias provides users with essentially infinite amount of storage. Anyone can create any amount of accounts on Twitter and Facebook, and with enough accounts one could potentially store all their data using such a filesystem. The thesis explores the worth to use a free filesystem that is potentially slower than one that costs a few dollars per month. Further, is it ethically defendable to use such a system?

1.5 Goals

The project aims to create a secure, mountable filesystem which stores its data for online by taking advantage of the free storage given to users. This can be split into the following subgoals;

1. to create a mountable filesystem where files can be stored, read and deleted
2. for the system to be secure in the sense that even with access to the uploaded files and the software, the data is not readable without the correct encryption key
3. to analyze the throughput of the filesystem and compare to commercial distributed filesystems

1.6 Research Methodology

The project will implement the filesystem using C++ and the library FUSE[2] for filesystem operations. Analysis and comparisons will be done by saving, reading and deleting files on FFS and competitive online filesystems such as Google Drive. The analysis with be with aspect to price, the speed of file operations, and other relevant metrics.

1.7 Delimitations

Due to limitations in time and its unnecessity for a woking filesystem, some features found in other filesystems are not being implemented in FFS. This includes for instance file access control and symbolic links. The reasoning is that we only want to present the possibility of creating a filesystem with a different storage medium and while these features are useful in a regular filesystem, FFS will only aim to be implemented as a minimalistic filesystem.

1.8 Structure of the thesis

Chapter 2 presents theoretical background information of filesystems and the basis of FFS while Chapter 3 mentions and analyses related work. Chapter 4 describes how the implementation and the choices made for the system, along with the analysis methodology. Following, Chapter 5 presents the results of the analysis and Chapter 6 discusses the findings and other aspects of the work. Lastly, Chapter 7 will finalize the conclusion of the thesis and discuss potential future work.

Chapter 2

BACKGROUND

2.1 Filesystems

Filesystems are used to store data on for instance a hard drive of a computer in the cloud. Google Drive is a filesystem that enables users to save their data online up to 15 GB for free[3] using their clusters of distributed storage devices, meaning that the data is saved on their servers which can be located wherever[4]. Paying customers can achieve a higher amount of storage using the service.

A Unix filesystem uses a data structure called an *inode*. An inode keeps track of the metadata for the files in the filesystem, and a directory simply contains the file names and each file or directory's inode id. Using a lookup, the system can then learn about the file - for instance where it is located and how big it is as can be seen in Figure 2.1 Each inode entry can contain any number of metadata information that might be relevant for the system, such as creation time and last updated.

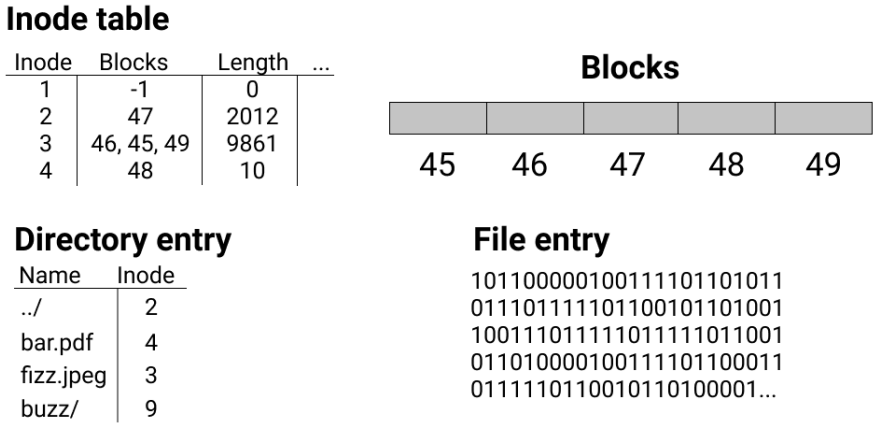

Figure 2.1: Basic structure of inode based filesystem

Looking at the 4 main file systems of windows, they all have many, sometimes different, functionalities such as links and named streams as well limitations such as a

defined theoretical maximum file size[5]. This is set to 16 exbibytes for NTFS, exFAT, and UDF, and for FAT32 it is set to 4 gigabytes.

2.1.1 Image structures

Different file types has different protocols and definitions of how they should be encoded and decoded, for instance a JPEG and a PNG can display similar content but the data they store is different. At the lowest level all files consists of a string of binary digits no matter the file type. If one would represent an arbitrary file of X bytes, each byte (0x00 - 0xFF) can be represented as a character such as the ASCII keyset and we can therefore represent this file as X different characters. Using the same set of characters for encoding and decoding we can get a symmetric relation for representing a file as a string of characters.

This string of X bytes can also be used as the data in an image. An image can be abstracted as a $h * w$ matrix, where each element is a pixel. Each pixel represents a color which can be represented as a number. One can therefore imagine that we can use this string of X bytes to assign colors in this color matrix by assigning the first Y bytes as the first pixels color, the next Y bytes as the following pixels color and so forth. The value of Y depends on the color type and depth of the image, for simplicity we can imagine an 8-bit RGB value, i.e. Y is 3 bytes (1 byte per Red, Green and Blue). This means that X bytes of data can be represented as

$$ceil(\frac{X}{3})$$

pixels, where *ceil* rounds a float to the closest larger integer. For a file of 1Mb, i.e. $X = 1'000'000$ we need $333'334$ pixels. The values of h and w are arbitrary but if we for instance want a square image we can set $h = w = 578$ which means that there will be $334'084$ pixels in total, and the remaining 750 pixels will just be fillers to make the image a reasonable size. We could however choose $h = 1$ and $w = 333'334$ which would mean a very wide image but would not require filler pixels.

2.2 Twitter

Twitter is a micro-blog online where users can sign up for a free account and create public posts (tweets) using text, images, and videos. Text posts are limited to 280 characters while images can be up to 5 Mb and videos up to 512 Mb[6]. There is also a possibility to send private messages to other accounts, where each message can contain up to 10'000 characters and the same limitations on files. It is also possible to create threads of Twitter posts where multiple tweets can be associated in a chronological order.

2.3 Threats

To consider a filesystem secure it is important to imagine different potential adversaries who might attack the system. Considering that FFS has no real control of the data stored on the different services, all the data must be considered to be stored in an insecure system. Even if we could hide the posts made on for instance Twitter by making the profile private, we must still consider that Twitter could be an adversary or that they could potentially give out information such as tweets or direct messages to entities such as the police. In fact, Twitter's privacy policy mentions that they may share, disclose and preserve personal information and content posted on the service, even after account deletion for up to 18 months[7]. Therefore the data stored must always be unreadable without the correct authentication. We assume that an adversary has access to all knowledge about FFS, including how the data is converted, encrypted, and posted. There are multiple secure ways of encrypting data, including AES which is one of the faster and more secure encryption algorithms[8]. However, even though the data is encrypted, other properties such as your IP address can be compromised which can expose the user's identity. This problem is however not addressed in FFS but is something for future work.

Other than adversaries for just FFS, we might also imagine that the underlying services might receive attacks that can potentially harm the security of the system or even have it go offline indefinitely. One solution is to use redundancy - by duplicating

the data over multiple services we can more confidently believe that our data will be accessible as all services will probably not go offline.

Chapter 3

RELATED WORK

3.1 Steganography and deniable filesystems

Steganography is the art of hiding information in plain sight, and has been around for ages. Today, a major part of steganography is hiding malicious code in for instance images, called stegomalware. Stegomalware is an increasing problem and in a sample set over 40% of real-life stegomalware was found in digital images[9]. While FFS does will not include malicious code in its images, this stegomalware problem flourish the development of detection techniques of steganography in for instance social medias, and it is of high importance and well researched.

Twitter has been exposed to allowing steganographic images that contains any type of file easily[10]. David Buchanan created a simple python script of only 100 rows of code that can encode zip-files, mp3-files and really any file imaginable in an image of the user's choosing[11]. He presents multiple examples of this technique on his Twitter profile[1]. The fact that the images available for the publics eye is evidence that Twitters steganography detection software might not be perfect. However ot os also possible that Twitter has chosen to not remove these posts

A steganographic, or deniable, filesystem is a system that does not expose files stored on this system without credentials - neither how many files are stored, their sizes, their content, or even if there exist any files on the filesystem[12]. This is also known as a rubberhose filesystem because of the characteristic that the data really only can be proven with the correct encryption key which only is accessible if the person is tortured and beaten with a rubber-hose because of its simplicity compared to the complexity of breaking the key by computational techniques.

[1] https://twitter.com/David3141593

3.2 Cryptography

Some papers choose to invent their own encryption methods rather than using established standards. Chuman, Sirichotedumrong, and Kiya proposes a scrambling-based encryption scheme for images that split the picture into multiple rectangular blocks that are randomly rotated and inverted, both horizontally and vertically, along with shuffling of the color components[13]. This is used to demonstrate the security and integrity of images sent over unsecure channels. In fact, the paper uses Twitter and Facebook to exhibit this. Despite its improvement of items such as bitsteam compliance, due to its well proven security FFS will use AES as its encryption method.

3.3 Related filesystems

Peters created a deniable filesystem using a log-based structure in 2014[12]. The filesystem of my project could be seen as a deniable system in the sense that the data is not stored on the device, and if the filesystem is not mounted it could be hard to prove that the user has data, even if they for instance would find the Twitter account. This was also developed using FUSE[2] which I also will be using.

Zadok, Badulescu, and Shender created Cryptfs, a stackable Vnode filesystem that encrypted the underlying, potentially unencrypted, filesystem[14]. By making the filesystem stackable, any layer can be added on top of any other, and the abstraction occurs by each Vnode layer communicating with the one beneath. There's potential to further stack through continuing layers by using tools such as FiST[15].

Chapter 4

METHOD

4.1 FFS

The product of this thesis is the Fejk File System (FFS) which uses online services to store the data but behaves like a mountable disk for the users. The file system will however be very basic and not support all functionalities that other systems do such as links. The reasoning is that these behaviors are not required for a useable system, and when comparing the system to distributed filesystems such as Google Drive, they often do not support his either.

Figure 4.1 describes the basic outline of FFS which is based on the idea of inode filesystems. Instead of an inode pointing to specific blocks in a disk, the inodes of FFS will instead point keep track of the id numbers of the posts to online services where the file is located.

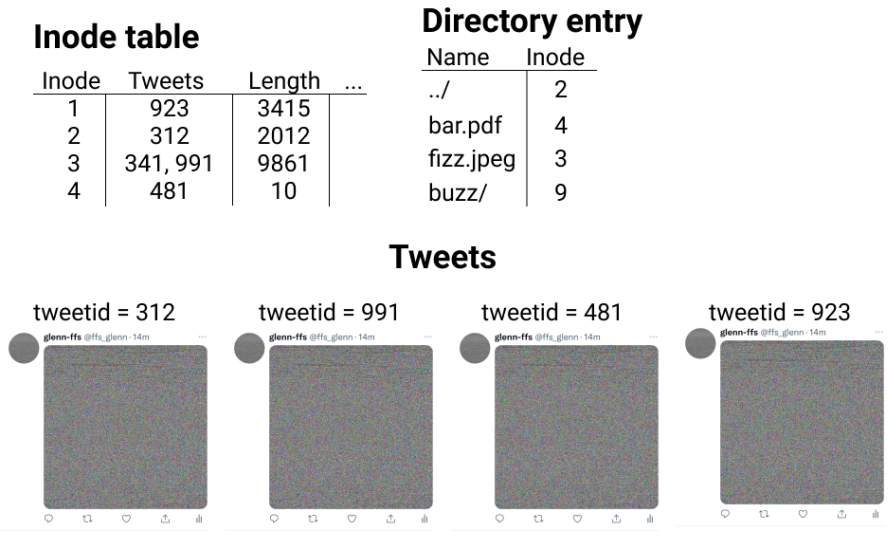

Figure 4.1: Basic structure of FFS inode-based structure

Chapter 5

RESULTS AND ANALYSIS

Chapter 6

DISCUSSION

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Future work

Bibliography

[1] *Mobile @Scale London Recap - Engineering at Meta*. URL: https://engineering.fb.com/2016/03/29/android/mobile-scale-london-recap/.

[2] *Libfuse*. libfuse, Oct. 26, 2021. URL: https://github.com/libfuse/libfuse (visited on 10/26/2021).

[3] *Cloud Storage for Work and Home – Google Drive*. URL: https://www.google.com/intl/sv/drive/ (visited on 10/26/2021).

[4] *Distributed Storage: What's Inside Amazon S3?* Cloudian. URL: https://cloudian.com/guides/data-backup/distributed-storage/ (visited on 10/26/2021).

[5] mikben. *File System Functionality Comparison - Win32 Apps*. URL: https://docs.microsoft.com/en-us/windows/win32/fileio/filesystem-functionality-comparison (visited on 02/07/2022).

[6] *Media Best Practices - Twitter*. URL: https://developer.twitter.com/en/docs/twitter-api/v1/media/upload-media/uploading-media/media-best-practices (visited on 10/26/2021).

[7] *Privacy Policy*. URL: https://twitter.com/en/privacy (visited on 02/15/2022).

[8] Dr Prerna Mahajan and Abhishek Sachdeva. "A Study of Encryption Algorithms AES, DES and RSA for Security". In: *Global Journal of Computer Science and Technology* (Dec. 7, 2013). ISSN: 0975-4172. URL: https://computerresearch.org/index.php/computer/article/view/272 (visited on 02/07/2022).

[9] *SIMARGL: Stegware Primer, Part 1*. URL: https://cuing.eu/blog/technical/simargl-stegware-primer-part-1 (visited on 02/09/2022).

[10] *Twitter Images Can Be Abused to Hide ZIP, MP3 Files — Here's How.* URL: https://www.bleepingcomputer.com/news/security/twitter-images-can-be-abused-to-hide-zip-mp3-files-heres-how/ (visited on 02/09/2022).

[11] David Buchanan. *Tweetable-Polyglot-Png.* Feb. 9, 2022. URL: https://github.com/DavidBuchanan314/tweetable-polyglot-png (visited on 02/09/2022).

[12] Timothy M Peters. "DEFY: A Deniable File System for Flash Memory". San Luis Obispo, California: California Polytechnic State University, June 1, 2014. DOI: 10.15368/theses.2014.76. URL: http://digitalcommons.calpoly.edu/theses/1230 (visited on 10/19/2021).

[13] Tatsuya Chuman, Warit Sirichotedumrong, and Hitoshi Kiya. "Encryption-Then-Compression Systems Using Grayscale-Based Image Encryption for JPEG Images". In: *IEEE Transactions on Information Forensics and Security* 14.6 (June 2019), pp. 1515–1525. ISSN: 1556-6021. DOI: 10.1109/TIFS.2018.2881677.

[14] Erez Zadok, Ion Badulescu, and Alex Shender. "Cryptfs: A Stackable Vnode Level Encryption File System". In: (), p. 14.

[15] *FiST: Stackable File System Language and Templates.* URL: https://www.filesystems.org/ (visited on 02/02/2022).