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 on-
line 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

APPENDICES

LIST OF TABLES

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/sending an encrypted version as one or more posts/private messages on these services. The goal is to achieve storage of data on the same scale as the free accounts on online storage services such as Google Drive where users can store up to 15Gb of files. Accomplishing this would mean that one can store more data than that for free using this new filesystem. The data posted will be encrypted and not be comprehensible by anyone who would stumble upon a post, just like if anyone would analyze a regular encrypted filesystem on a disk.

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 by allowing users to post text and files, almost unmonitored. The performance will however be analyzed and compared to existing alternatives such as Google Drive.

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 (**CITATION NEEDED?**). 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(**COMPARE iPHONE FROM LIKE 10 YEARS AGO AND TODAY - STORAGE AVAILIABLE, INCREASED. LOWEST TIER VS HIGHEST TIER**). 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 (**CITE??**). 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 (**FIND SOMETHING THAT USERS NEVER DOWNGRADES THEIR STORAGE - MUST EXIST**). 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 (**PROBABLY?? IS THAT GOOD ENOUGH FOR A THESIS?**). (**MENTION HERE ABOUT POTENTIAL ANOMALY DETECTION?? OR LATER?+**). 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 (**RIGHT?? I COULD NOT FIND ANYTHING WITH A QUICK GOOGLE SEARCH. LOOK AT TOS?**)

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[2] using their clusters of distributed storage devices, meaning that the data is saved on their servers which can be located wherever[3]. Paying customers can achieve a higher amount of storage using the service.

A 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[4]. This is useful if for example one is to be exposed to an audit of their data by a totalitarian regime where they don't even want to disclose that they have data.

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 (**CITATION NEEDED**). Each inode entry can contain any number of metadata information that might be relevant for the system, such as creation time and last updated.

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.
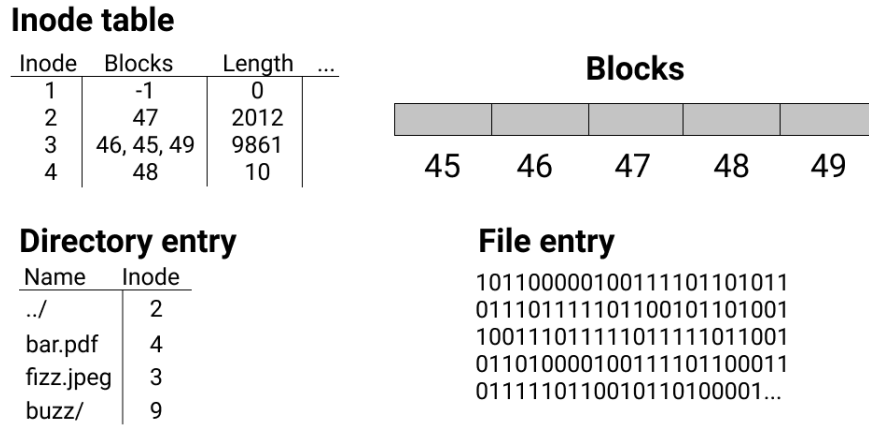
Inode table

Inode	Blocks	Length	...
1	-1	0	
2	47	2012	
3	46, 45, 49	9861	
4	48	10	

Blocks

45 46 47 48 49

Directory entry

Name	Inode
../	2
bar.pdf	4
fizz.jpeg	3
buzz/	9

File entry

10110000010011110110101\
0111011111011001011010010\
10011101111101111101100\
011010000100111101100011\
0111110110010110100001...

Figure 2.1: Basic structure of inode based filesystem

2.2 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, and 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[6].

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.

2.3   Twitter

Twitter is a micro-blog online where users can sign up for a free account and create public posts 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[7]. 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. If one would represent an arbitrary file of X bytes, each byte (0x00 - 0xFF) can be represented as a character 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 text can theoretically be posted on for instance Twitter, as long as the size is smaller than 280 or 10'000 bytes depending on if we would post a public post or a private message.

2.4   Related Work

Peters created a deniable filesystem using a log-based structure in 2014[4]. 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[8] which I also will be using.

Zadok, Badulescu, and Shender created Cryptfs, a stackable Vnode filesystem that encrypted the underlying, potentially unencrypted, filesystem[9]. 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[10].

Chapter 3

METHOD

3.1 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 3.1 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.
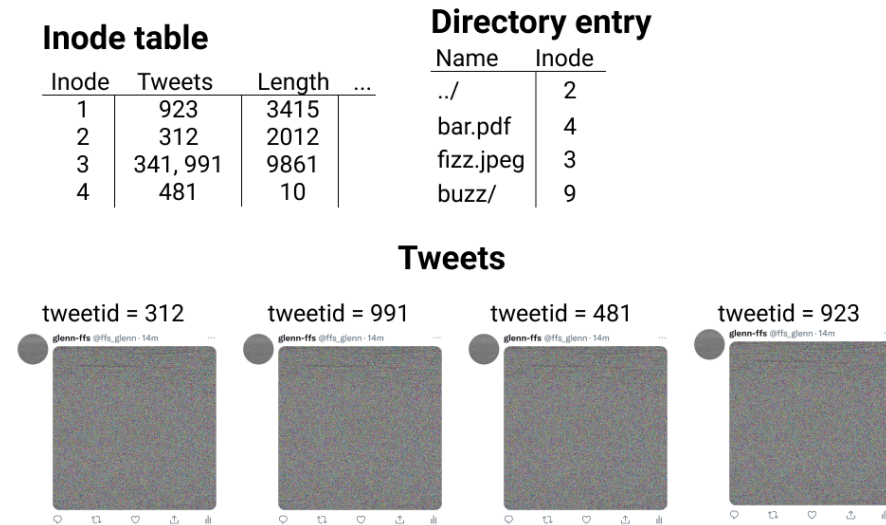
Inode table

Inode	Tweets	Length	...
1	923	3415	
2	312	2012	
3	341, 991	9861	
4	481	10	

Directory entry

Name	Inode
../	2
bar.pdf	4
fizz.jpeg	3
buzz/	9

Tweets

tweetid = 312   tweetid = 991   tweetid = 481   tweetid = 923

Figure 3.1: Basic structure of FFS inode-based structure

Chapter 4

RESULTS AND ANALYSIS

Chapter 5

DISCUSSION

Chapter 6

CONCLUSIONS AND FUTURE WORK

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

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

[4] 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).

[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] 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).

[7] *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).

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

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

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