Evaluation of Deduplication in IPFS

Deep Phuyal
Department of Computer Science
University of Mississippi
Oxford, USA
dphuyal@go.olemiss.edu

***Abstract*—Peer-to-peer file sharing system has evolved over time. The Interplanetary File System (IPFS) is the one with the ambitious goal to replace the existing web. IPFS has borrowed several techniques from other peer-to-peer file sharing system and version control system. This paper evaluates one of the features of IPFS; deduplication.**

Keywords*— *IPFS, peer-to-peer, filesharing system

I. Introduction

In the past decade, the research in constructing peer-to-peer distributed file system has significantly grown. In the peer-to-peer network, each peer is equal to the other peers i.e., no nodes are privileged. They can share information directly among themselves instead of concentrated at a single server. There are various peer-to-peer systems such as Napster, Kazaa, and BitTorrent that has gained a lot of popularity for large file distribution systems. Even though the systems can support over millions of users simultaneously, they were not designed as infrastructure to build upon [1,2]. However, IPFS was able to successfully use those infrastructure (especially BitTorrent) to build a filesharing system that aims to fundamentally change the way information is distributed [3].

IPFS is a protocol and peer-to-peer network for storing and sharing data in a distributed file system that can work across places as disconnected or as far apart as planets. IPFS is an open-source project created by Juan Benet and Protocol Labs. It attempts to solve the shortcomings of the client-server model and HTTP web through a novel peer-to-peer file sharing system. IPFS is a synthesis of successful ideas from several new and existing peer-to-peer systems, such as Distributed Hash Tables (DHTs), BitTorrent, Git, and Self-Certified Filesystems (SFS). Git, a distributed version control system has influenced distributed filesystem design and IPFS is no exception. Git's Merkle DAG (Directed Acyclic Graph) data model enables powerful file distribution strategies. As mentioned in IPFS white paper, the central IPFS principle is modeling all data as part of the same Merkle DAG [2].

IPFS protocol is divided into a stack of sub-protocols; identities, network, routing, exchange, objects, files, and naming. Identities manage node identity generation and verification. Nodes are identified by a cryptographic hash of a public key created using S/Kademlia's static crypto puzzle called a NodeId [2]. Network subprotocol manages connections to other peers using various network protocols. Routing maintains information to locate specific peers and objects. Exchange subprotocol uses block exchange protocol called BitSwap to govern block distribution. Objects is a Merkle DAG of content-addressed immutable objects with links [2]. Files – versioned file system similar to Git. Naming – IPFS uses naming scheme from Self-certifying File (SFS) to create self-certified names [2].

Merkle DAG is a directed acyclic graph where each node has a unique identifier, which is the outcome of hashing the node's contents using a cryptographic hash function like SHA256. Merkle DAG is important to IPFS because it provides following three useful properties:

1. Content Addressing: content has a unique identifier that is the cryptographic hash of the file. That is instead of an identifier that addresses by location in current HTTP web, IPFS addresses it by its cryptographic hash (or content).
2. Tamper Resistance: data is verified with its checksum. So, if the hash changes, IPFS detects the data is tampered with [2].
3. Deduplication: files with the same content is only store once and cannot be duplicated.

In this project, we present an evaluation of deduplication mechanism of IPFS against various test parameters. The rest of the paper is summarized as follows: Section II provides an overview of project, Section III discusses testing methodology, Section IV shows deduplication evaluation results, and Section V discusses the future works and conclusions.

II. Project Overview

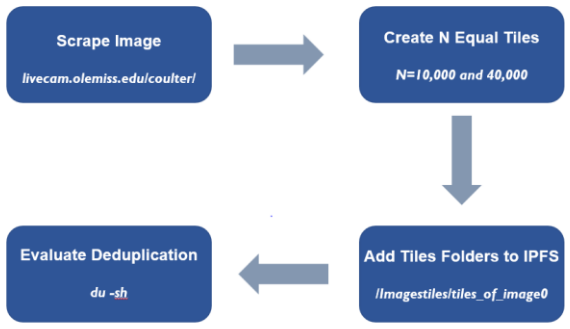
Scrape Image
livecam.olemiss.edu/coulter/
Create N Equal Tiles
N=10,000 and 40,000
Add Tiles Folders to IPFS
/Imagestiles/tiles_of_image0
Evaluate Deduplication
du -sh

Fig. 1. Flow diagram of the project

A. *Image Scrapping*

The initial step of this project is the image scraping from the web. To scrape the images, a python script was written using Beautiful Soup library. Beautiful Soup is a popular python library widely used to scrape information from web pages. One of the many live cams on campus at the University of Mississippi is used to scrape out the live images. The one used in this project is: *https://livecam.olemiss.edu/coulter/* After successful execution of the script, 1000 images (similar to the figure 2) were scraped out of the above-mentioned live cam at 5 seconds and 2 seconds interval between each image and saved to an individual folder. Total time taken to scrape 1000 images at 5 seconds interval was roughly 40 minutes and around 30 minutes for 1000 images at 2 seconds interval.

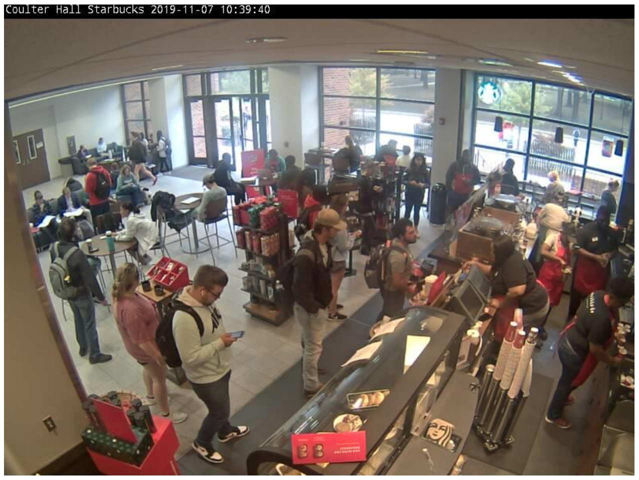

Fig. 2. Sample image scraped from the live cam

B. *Tiles Creation*

The next step in the project is creation of equal number of tiles of an image. To achieve that, a python script was written using Image module of Python Imaging Library (PIL) which provides the python interpreter with image editing capabilities. The script sliced each image to a N number of equal tiles and automatically saved all the tiles of that image to a folder at a specified path. Thus, 1000 folders were created with desired number of tiles at a user-specified path. For instance: folder named slices_of_image0 could contain 40,000 (N) equal slices or titles of image 0.jpg. For this project two N values were used; 10,000 and 40,000. The reason behind this is discussed in section [x].

C. *Adding Folders to IPFS*

Once the tiles are created and saved to a folder, the folder is ready to be added to the IPFS network. Chameleon Cloud nodes used in this project required installation of IPFS in their system. The installation process of IPFS is simple and takes only few minutes. Since IPFS was used for the first time on those nodes, initialization of repository was required using ipfs init command. IPFS stores all its settings and the internal data in that repository (or directory). Folders in IPFS is added using ipfs add -r <path> command where <path> is the path to a file/folder to be added to IPFS and -r to recursively add directory paths.

D. *Deduplication Evaluation*

When the folders are uploaded to the IPFS, they are chunked by IPFS and stored in local cache folder (.ipfs). Using disk usage (du) command, size of the tiles folders added to .ipfs can be evaluated. By comparing the size of .ipfs folder to the original ImagesTiles folder, the amount of deduplication is evaluated.

III. TESTING METHODOLOGY

There are several cloud computing providers such as Amazon, Google, Microsoft. However, Chameleon cloud resources were used for this project because it is available to computer science researchers and students for free. Chameleon has different flavors of instances like m1.tiny, m1.medium, m1.large. For this project, two instances of m1.large was used which came with a 8GB RAM, 4 vCPUs, and 80GB of disk storage capacity. Both the instances did an excellent job of scrapping 1000 images from the live cam at 2 seconds interval and 5 seconds interval. However, it took them quite some time to slice those images into 10,000 and 40,000 tiles. Due to which, only 100 images (of both 2 seconds and 5 seconds interval) were used on this project. Also, adding each folder of 40,000 tiles of 100 images into IPFS took approximately 22 hours of computation image because the total size of those folders was 15GB discussed in section IV. It could have taken days to add 40,000 tiles of 1,000 images to IPFS, so using only 100 images made more sense.

	No. of tiles	**Original folder size (MB)**	**.ipfs folder size (MB)**	**Deduplication (%)**
RGB	10,000	78	73	6.41
	40,000	312	266	14.74
B&W	10,000	78	70	10.25
	40,000	312	228	26.92

Table. 1. Deduplication results of two images scraped at 5 seconds interval.

Before evaluating deduplications on 100 images, few tests on two images were performed to see results of both RGB (color) and black & white images. As shown in the table 1, 10,000 tiles of two black & white images captured at 5 seconds interval had higher deduplication percentage compared to the 10,000 tiles of two RGB images. Similarly, 40,000 tiles of two black & white images had significantly higher percentage of deduplication than 40,000 tiles of two RGB images.

Similar tests were ran on two images scraped at 2 seconds interval. As shown in the table 2 above, black & white images tiles (both 10,000 and 40,000) had higher percentage of

	No. of tiles	Original folder size (MB)	.ipfs folder size (MB)	Deduplication (%)
RGB	10,000	78	73	6.41
	40,000	312	258	17.30
B&W	10,000	78	69	11.53
	40,000	312	219	29.80

deduplication compared to RGB images scraped at 2 seconds interval.

Table. 2. Deduplication results of two images scraped at 2 seconds interval.

Since black & white images tiles had better deduplication results than RGB images tiles, black & white tiles were used to calculate deduplication on 100 images. Another reason to choose black & white images tiles was the computation time. To add 40,000 tiles of two black & white images (scraped at 2 seconds interval) to IPFS took approximately 40 minutes on an instance in the cloud and approximately 15 minutes to add 10,000 tiles of two black & white images.

IV. Deduplication Evaluation Result

After the decision to use black & white images was backed by the numbers as discussed in section III, python script to create 40,000 tiles of all 1,000 images scraped at 5 seconds interval was ran. However, the total size of those folders exceeded the disk storage allocated to the instance running the script, it terminated after creating 40,000 tiles of close to 500 images. Since the total size of the tiles folder of 100 images was 15GB, only 100 images were added to the IPFS using ipfs add -r command. As shown in the table 3, it took approximately 22 hours to add all the folders to IPFS. The size of .ipfs folder after the completion of addition process was 8.6GB. IPFS identified 6.4GB of duplicated tiles of 100 images, which is 42.46%. Likewise, the total size of all the folders with 10,000 tiles of 100 images was 3.8GB. Adding them to IPFS took approximately 6 hours. After the successful completion, size of .ipfs folder was 3.0GB. IPFS identified 0.8GB of duplicated tiles of 100 images, which is 21.05%. As of now, there are no benchmarks to compare that number with, but it is an excellent number

No. of tiles	Original folder size (GB)	.ipfs folder size (GB)	Execution time (approx.)	Deduplication (%)
10,000	3.8	3.0	6 hours	21.05
40,000	15	8.6	22 hours	42.46

Table. 3. Deduplication results of 100 images scraped at 5 seconds interval

Similarly, folders containing 10,000 tiles of each 100 images scraped at 2 seconds interval were created using the python script. The total size of those folders was 3.8GB like the 100 images scraped at 5 seconds interval. As shown in the table 4, it took approximately 6 hours to add all the folders to IPFS. The size of .ipfs folder after addition was 2.8GB. The deduplication percentage in this case was 26.31%, which is higher than the 10,000 tiles folder of 100 images scraped at 5 seconds.

No. of tiles	Original folder size (GB)	.ipfs folder size (GB)	Execution time (approx.)	Deduplication (%)
10,000	3.8	2.8	6 hours	26.31

Table. 4. Deduplication results of 100 images scraped at 2 seconds interval

V. Conclusion And Future Work

Although the great deduplication results were achieved for 10,000 tiles and 40,000 tiles of 100 images scraped at 5 seconds interval, I wish I had more time to test the deduplication for 40,000 tiles of 100 images scraped at 2 seconds. The deduplication result for 10,000 tiles for 2 seconds interval 100 images could have been better than 5 seconds interval 100 images because of lesser movements of students and Starbucks workers. Although our naked cannot detect the change, but a slight change in the light (maybe from opening the door) could change the bytes position of the image. Due to this reason, Dr. Rhodes suggested to use bit shifting technique to yield higher deduplication results. However, I was not able to implement that technique in my project. That would be the addressed in the future. Overall, the results were impressive and exceeded my expectations.

Acknowledgment

I would like to thank Dr. Philip Rhodes for his assistance throughout the project and constructive comments that helped me yield better results. It was a great learning experience to work on a peer-to-peer distributed file sharing system and cloud computing environment. I certainly learned plethora of things during the development of this project. Dr. Rhodes was exceptionally helpful setting up the instances on Chameleon Cloud and I would like to thank him again for that.

References

[1] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, “A survey of peer-to-peer storage techniques for distributed file systems,” International Conference on Information Technology: Coding and Computing (ITCC05) - Volume II, 2005

[2] J. Benet, “IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3),” https://ipfs.io/. [Online]. Available: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf. [Accessed: 13-Dec-2019]

[3] K. Addaquay, “A Beginner's Guide to IPFS,” By. [Online]. Available: https://hackernoon.com/a-beginners-guide-to-ipfs-20673fedd3f. [Accessed: 13-Dec-2019]