卒業論文 2017 年度 (平成 29 年)

Immutability of Open Data through Distributed Storage

慶應義塾大学　環境情報学部
ローランド　リチャード

卒業論文要旨 - 2017 年度 (平成 29 年度)

分散ネットワークを用いたオープンデータの半永久保存

時系列データが用いられる科学的調査において、誰でも使えるように公開されてオープンデータは有益である. 常時アクセス可能になっていることで科学者の [仕事ができる]. 通常、これらのデータが保存されているレポジトリは中央化されているかいくつかのサーバーにてホストされている.

しかし、オープンデータのアクセス可能性は技術や政治の要因を絡めた理由によって遮断されることもある. これらのデータが使用できなくなった場合、調査へ大きな障壁になる.

このような懸念を解消するべく、この研究ではオープンデータのホスティングを分散された手法に移行する. 分散型ホスティングにより無数の場所にてデータが常時複製されるようになり、障害の元となる出来事や人物がオープンデータへのアクセスを遮断することを防げるようになる. このようなモデルにより、一箇所のサーバーがデータを提供できなくなった場合、他のサーバーが代わりにデータを提供することができる.

以上の提案を実装するため、この研究では InterPlanetary File System (IPFS) を活用する。IPFS はデータを分散型ネットワークにホスティングをするため、データが常時アクセス可能となり、データ配送を効率化する。同時に、Bitcoin ブロックチェーンにタイムスタンプを書き込み、データが特定の日時に既存していたことを証明する。

本研究の成果により、分散型ホスティングでも速度面や真正性において十分な効果が発揮されることが判明し、さらに中央型ホスティングにおける信頼悪用 の懸念を払拭することができた。

キーワード:
1. 分散ストレージ, 2. ブロックチェーン, 3. オープンデータ, 4. ピアーツーピアー, 5. 分散ネットワー

慶應義塾大学 総合政策学部
ローランド　リチャード

Abstract of Bachelor's Thesis - Academic Year 2017

Immutability of Open Data through Distributed Storage

Open Data has been useful in scientific research, especially when historical data is required. Its constant availability allows scientists to use them to assist their reseatch at any time they desire. These data are often hosted in repositories that are either centralized in one location or duplicated in several locations.

However, the availability of open data can be threatened for various reasons, include those of technical or political nature. Should these data become unavailable, it will hamper research reliant on such data.

To resolve the concerns of potential obstruction of accessibility, this paper proposes to mitigate such risks by hosting open data in a distributed manner. Distributed hosting will prevent an event or actor to obstruct access to open data, as the data will be constantly duplicated in an indefinite number of locations. In this scheme, should one locaiton fail to serve requested data, the data can be accessed from another data.

As an approach to implement the proposal above, this research will use the InterPlanetary File System (IPFS). IPFS hosts data in its distributed network, which achieves constant availability and delivery efficiency. At the same time, we will use the Bitcoin blockchain to timestamp these data in order to prove their existence at given dates.

Based on the results of the experiment, IPFS can serve data with just as much utility as centralized methods, allowing data to be accessible almost as soon as it is uploaded.

Keywords :
1. Distributed Storage, 2. Blockchain, 3. Open Data, 4. Peer-to-peer, 5. Distributed Network

Keio University, Faculty of Enviornmen and Information Studies
Richard Rowland

目 次

図 目 次

表 目 次

第1章　Introduction

This chapter will highlight the cardinal theme of this research by explaining the issue and the proposed solution.

1.1　Development of Open Data

The availability of Open Data has been useful for scientific research. Open Data has some benefits, as its public nature allows external parties to examine its transparency and integrity.

Open Data has been published in numerous cases, including in science and government. These data are hosted by various kinds of organizations, including government, corporations, and non-profit organizations.

1.2　SAFECAST

One application of open data is SAFECAST. Established in response to the Fukushima nuclear disaster in Marh 11, 2011, SAFECAST provides data regarding radiation whereever its sensors are deployed, mainly in Japan.

1.3　Problem

One of the critical problems of open data is the neutrality of the hosting party, especially that pertaining to politics. The host needs to be trusted to provide the data reliably with integrity. In this setup, the host has to make sure the data is available all the time without making unauthorized changes to the data. If the host holds any motive to promote a certain perspective with data they host, the trust held publicly for the integrity of the data is compromised.

In a circumstance when a single host is entrusted to host data in a single repository, the host attempts to gain public trust by claiming neutrality, such as by abstaining from promoting any political views.

However, if the hosting party is a govermental organization, it may be compelled to give up on its neutrality under political pressure. For a recent example, the Envrionment

Protection Agency under the Trump administration has recinded information and data regarding climate change, as the new administration had denied it.

The risks relating to having a single repository can be somewhat alleviated by replicating the dataset into several repositories, which makes it more difficult to alter data retroactively. However, it is still difficult to make it completely incompromable against a powerful adversary.

1.4 Hypothesis

To mitigate the issues above, this research proposes to use distributed networks as an alternative method to host data. By employing P2P networks, it will become very difficult for an adversary to censor information, as the data will exist in innumerable locations.

1.5 Approach

This research will use and examine the efficacy of the InterPlanetary File System (IPFS). Open Dataset from SAFECAST will be hosted on this network.

1.6 Structure of this Thesis

Chapter 2 discusses the background and problems surrounding Open Data that has lead to this research.
Chapter 3 discusses the approaches taken to the problem of distributed databases.
Chapter 4 describes the core technologies utilized in the implementation for this research.
Chapter 5 describes the evaluation including the experiement and results.
Chapter 6 wraps up the research with a conclusion.

第2章　About this Research

This chapter will explain about the issues pertaining to Open Data as well as the technologies utilzied to solve such problems.

2.1　Open Data

Open Data is promoted under the notion that data should be made public and freely reusable, without any restrictions on patent, copyright, or any means of control as an intellectual property. According to the Open Definition, open data can be defined as such:

> "Open data is data that can be freely used, re-used and redistributed by anyone - subject only, at most, to the requirement to attribute and sharealike."[3]

The cirteria for the openness include:

- Availability and Accessibility
  the dataset should be accessible in its entireity at a reasonable cost, ideally via affordable means such as downloading over the internet. The data should also be made available in a convenient and modifiable format.

- Reuse and Redistribution
  the dataset should be made available udner terms that allow re-use and redistribution, such as mixing with other datasets

- Universal Participation
  anyone can be able to use, reuse, and redistribute, so that no personnel or group is excluded from its use. For instance, "non-commercial" licenses that exluded "commercial" usage should not be allowed.

The main reason behind promoting open data is interoperability. Interoperability means that various organizaiotns and systems can work together, in this case on different datasets.

Interoperability encourages the development of more complex systems by having componenets "plug together". With the ability to mix in more datasets, it enables innovation by making it easier to develop better ideas, insights, products, and such.

Open Data plays an important part in scientific research. For example, better access increase the rate of scientific discoveries.[4] Also, open data prevents "data rot", or conditions in which data from older scientific research becomes increasingly unavailable due to outdates data saving methods, e.g. floppy discs.[5][6]

2.1.1 SAFECAST

As an example of the use of Open Data, SAFECAST provides radiation data sampled from their network of sensors.[7] Formed in response to the Fukushima Earthquake and the following meltdown of the nuclear power plants on March 11, 2011, SAFECAST had been aiming to collect and provide reliable data on nuclear radiation. To enhance trust towards their activities and published data, SAFECAST implements a few measures. Their foremost feature is their openness, in which all their data, hardware, software and such are made public. This allows the public to examine their reliability and participate by contributing to their development. In addition, SAFECAST claims political neutrality and mostly refrains from affliation with external organizations, so that their decisions are not swayed by external forces.

Data Collection on SAFECAST

SAFECAST provides methods to collect and access data as openly as possible.

Anyone can contribute data to SAFECAST's dataset. The data format is in csv, which is open and non-proprietary. Typically, radiation is measured by geiger counters. SAFECAST has developed their own model of geiger counters called bGeigie, which is made open source and can be modified by anyone. The bGeigie contains the following sensors including the pancake Gieger and GPS.[8] It is also capable of logging data onto an SD card for a duration desired by its user. Later, the recorded data can be extracted from the SD card and uploaded to SAFECAST server.

Uploading data can be done through the SAFECAST API.[9] Contributors can either use a web site provided by SAFECAST that can be access via a browser, or through a command line interface. Here, we will use the web site version to explain how contributing data to SAFECAST works.

For more information about how to upload log data from a begeigie counter, please look at Appendix B.

2.2 Problems Facing Open Data Today

While the advocacy of Open Data has seen some success, problems that need to be addressed have arisen.

2.2.1 Political Threat

While Open Data provides useful benefits to the scientific community and society as a whole, its availability can be threatened by political forces. If the dataset is hosted by a group, manegerial decisions in the orgnization may compel the removal of such dataset from the public.

One recent event that serves as a notable example is the removal of dataset regarding the climate by the Enviornment Protection Agency (EPA) of the United States Federal Government. Under the new administration of President Donald Trump, the EPA had been assigned Scott Pruitt as its director. As sceptics of climate change, Trump and Pruitt directed the EPA to remove climate data from its official website, thereby making it no longer accessible to the general public. The unavailabilty will be detrimental to scientists and researchers who had depended upon this dataset to conduct their research.[10]

2.2.2 Proof of Existence

Given how open data has been mostly hosted through centralized means, it was not common to provide, along with the publication of data, the proof of existence of files either on first-party or third-party outlets. This left almost no way to examine the existence of files on the claimed dates.

Caching websites like the the Wayback Machine provided by the Internet Archive[**?**] can somewhat provide a way to examine a proof of existence by scraping these websites as they crawl. However, crawling is usually incomplete, failing to download resources like images or scripts, and can take a lot of storage space to save these resources.Additionally, caching websites are centralized, meaning they are susseptible to inappropriate modifications by malicious actors.

第3章 Approach taken in this research

This chapter will cover the problems facing open data currently and approaches towards them.

3.1 Defining the Problem

For open data to be effective, it needs to be hosted in a way that is highly available, ideally with no downtime. However, availability may be compromsied given technical or political reasons. To accomodate these problems, it is necessary to come up with another approach that maximizes availabity by removing technical obstacles while making it difficult to censor for political motivations through archtetural means.

In this research, the problem lies on the limited availability of open data in the currently available means of hosting them.

3.2 Dissecting the problem

Since open datasets exist in a form of a database, it is useful to understand the theory of databases regarding how they are stored and hosted. Namely, the is the CAP theorem[11] postulates the difficultly of hosting a decentralized database in its acronymn[12]:

- Consistency
  This refers to how transactions remain valid from one database to another. All nodes see the same data at the same time.

- Availability
  This refers to whether the data can be retrieved at given times. Ideally, the data should be available on demand 24/7. That means that node failures do not prevent remaining nodes from continuing their operation.

- Partition-Tolerance
  This refers to how data service can continue in case the node cluster serving them breaks up in events like data inconsistencies. Ideally, data can be continued to

be served even when node clusters divide. In this case, the system continues its operation in spite of message loss attributed to network or node failure.

The CAP theorem proposes that not all three criterias can be achieved simlutaneously, but instead only two can be satisfied at the same time.

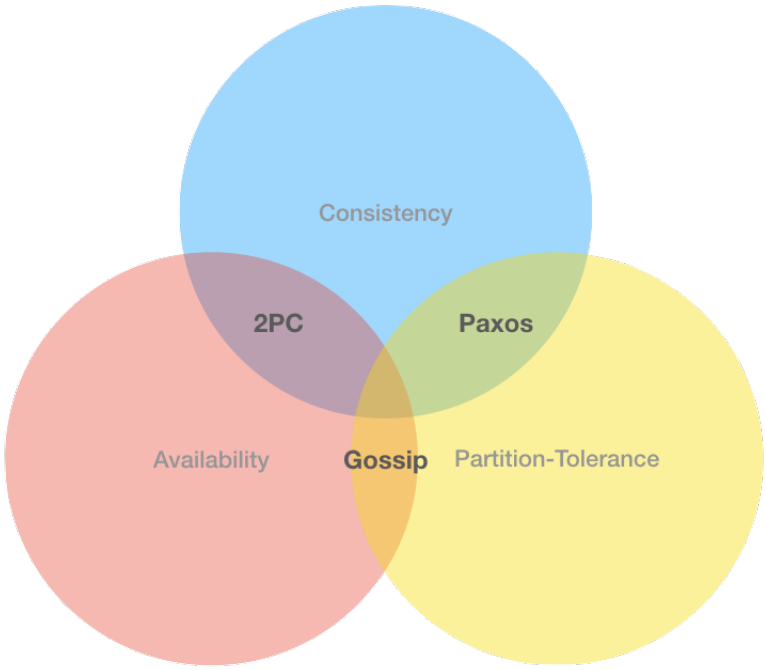

図 3.1: Design choices based on the CAP theorem

This contrainst calls for a compromise when designing networked systems. Typically, systems are built to be distributed to mitigate risks of a singular system; if there is a failure in the network or infrastructure with that system, the entire system fails. It is expected that the system hold some partition tolerance, as the system is likely to be built in a distributed manner. By this decision, the factor of Partition-Tolerance will be already picked from the choices prsented by the CAP theorem.

Therefore, system designers are usually confronted with the choice of taking either of the remaining two design factors, availability or consistency. If the system designer picks availability, then the system will continue to serve data, but without the guarantee of consistency. If he prioritizes consistency, then the sytem will serve data that is agreed upon in all datasets, but a failure in one node will break its consistency, and will be unavailable until the node recovers and successfully recovers to reconnect and syncronize data with the other nodes.

With such limitations, database designers either have the choice of choosing databases

management system designed for CP (Consistency and Partition-tolerance) or AP (Availability and Partition-tolerance). Database management system of the respective categories include:

CP

- Chubby[13], Doozer[14] (Paxos)
- ZooKeeper[15] (Zab)
- Consul[16], etcd[17] (Raft)
- N/A (Viewstamped Replication)

AP

- Cassandra[18]
- Riak[19]
- Mongo[20]
- Couch[21]

For reference, should the sytem designers decide to overlook at Partition-tolerance and instead choose to implement the design factors of CA (Consistency and Availability), such system will feature strict quorum protocols like two-phase commits.

CA and CP systems both offer the same consistency model of strong consistency.

As for open datasets like SAFECAST, the properties dataset is historical, so past values are unlikely to change. At the same time, it is important for such dataset to be as available as possible, given its archival value. Therefore, it would be prudent to opt in for Availability over Consistency, and would prefer an AP database management system.

第4章　Implementation

This chapter will talk about the core technologies that enable the implementation of the system for this research.

The system design is shown in Image 4.1.

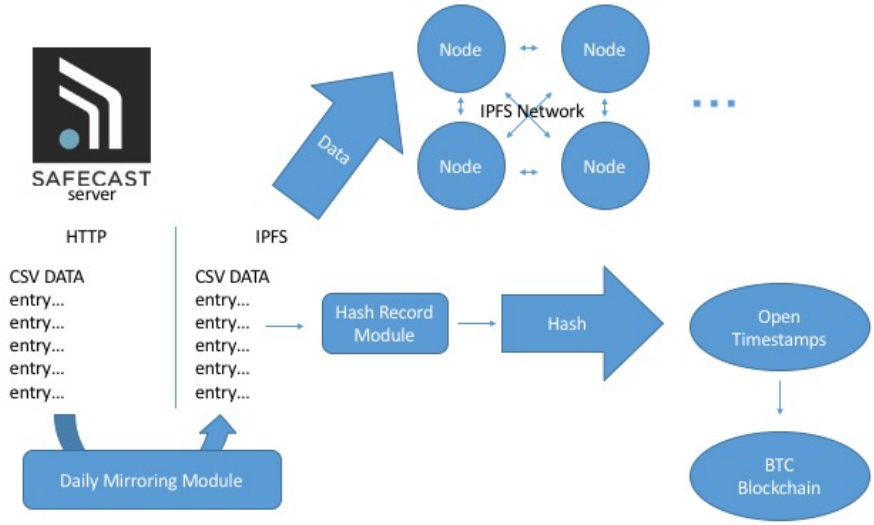

図 4.1: System Diagram

The system relies on several external components, including:

- IPFS
- Bitcoin
- OpenTimestamps

In this research, we designed a program to orchestrate the passing of data along among these components. In the intrest of illustrating how the overall system works, we will first look into the mechanism of these components.

4.1 IPFS

The InterPlanetary File System (IPFS) is a distributed file system that utilizes peer-to-peer (P2P) technologies.[22] It connects like a single BitTorrent swarm, exchanges data objects within one Git repository, and can be accessed in a manner likewise to the Web. IPFS combines a distributed hashtable, block exchange, and a self-certifying namespace. This architecture removes a single point of failure.

4.1.1 Background

This section reviews the technogies combined to realize IPFS.

DHT Distributed Hash Tables (DHTs) are used widely to coordinate and maintain metadata within P2P systems. It works like a dictonary service over a distrbuted network, with a data-store that pairs shared-key and the corresponding value.

Generally in DHTs, each node is identified by a chosen ID. Each value stored on the network is identified by a randomly distributed key of the same size. Every node has partial knowledge of the network by keeping a routing table of known node IDs and their IPs. The routing table is structured as a binary tree of node IDs. Also in every node, data is stored. To find the nodes globally responsible of storage for a key `H`, a node will contact the nodes in its table that are the closest to `H`. Requests include the key, while the response include values for `H` if available, IDs and IPs of nodes closest to `H` in the answering node's table.[23]

Popular DHT technologies include *Kademlia DHT*, *Coral DHT*, and *S/Kademlia DHT*.

Block Exchange - BitTorrent BitTorrent is a popular P2P filesharing system that coordinates netowrks of swarms or untrusted peers to cooperate in distributing portions of files among each other. The design on IPFS borrows ideas from BitTorrent including:

1. BitTorrent's "tit-for-tat" strategy that rewards nodes contributing to the network and punishes those who leech from other's resources
2. BitTorrent's tracking of file pieces, prioritizing rarest pieces first
3. PropShare's peer bandwidth allocation strategy that alleviates the vulnerabilities of BitTorrent's standard tit-for-tat strategy

Version Control Systems - Git Version Control Systems provide ways to track over changes in files and distribtue different versions efficiently. Git, a popular version control system, provides a useful Merkle Directed Acyclic Graph (DAG) object model that lends its utility towards distribution.

1. The following are represented accordingly
    - Immutable objects - `blob`
    - Directories - `tree`
    - Changes - `commit`
2. Objects are represented by their cryptographic hashes
3. Links to other objects are embedded, forming a Merkle DAG
4. Most versioning metadata are used as simple pointer references
5. Version changes only update references or add objects
6. Distributing version changes to other users simply requires transferring objects and updating remote references

Self-Certified Filesystems - SFS SFS promoted implementations of both (a) distributed trust chains, and (b) egalitarian shared global namespaces. SFS introduced a technique for building *Self-Certified Filesystems* by addressing remote filesystems using the scheme below:

```
/sfs/<Location>:<HostID>
```

where `Location` is the server network address, while

```
HostID = hash{public_key || Location}
```

This makes the *name* of the SFS file system certify its server. This allows users to verify the public key offered by the server, negotiate a shared secret, and secture all traffic. All SFS instances share a global namespace where name is allocated by cryptographic schemes without relying on any centralized body.

4.1.2 Design

IPFS combines ideas for existing P2P systems, including the above-mentioned DHT, BitTorrent, Git, and SFS. IPFS merges these in a simplified cohesive manner.

Identities Identities are formed by giving each node a `NodeID`, the cryptographic hash of a public-key. Nodes store their public and private key pair.

Network IPFS nodes communicate frequently with other nodes within the network. The network stack includes:

- Transport
  IPFS can use any transport protocol.
- Reliability
  IPFS can provide reliability should underlying networks lack them, by using uTP.
- Connectivity IPFS also utilizes the ICE NAT traversal techniques.
- Integrity
  Integrity of messages can be checked by a hash checksum.
- Authenticity
  Authenticity of messages can bec checked using HMAC with the sender's public key.

Routing IPFS nodes call for a reouting system that helps find (a) other peers' network addresses and (b) peers who can server particular objects. IPFS achives this using DHT. Small values ($\leqslant$ 1KB) are stored directly on the DGT, while larger values are stored as references, which are `NodeID` of peers who can serve the block.

Block Exchange IPFs enables data distribution by exchanging data blocks in a way inspired by BitTorrent: BitSwap. In BitSwap, peers are looking to obtain a set of blocks (`want_list`) and have another set of blocks to offer in exchange (`have_list`). As a feature differentiating from BitTorrent, BitSwap is not limited to the blocks in one torrent but can obtain any kinds of blocks regardless of what files those blocks comprise into.

Object Merkle DAG The DHT and BitSwap enables IPFS to orchestrate a massive P2P system for sotring and distributing blocks in a swift and robust manner. On top of these, IPFS builds a Merkle DAG that links objects with cryptographic hashes of targets embedded in the sources. The properties of Merkle DAGs include:

1. Content Addressing
   All content is uniquely identified by its checksum.
2. Tamper Resistance
   All content is verified by its checksum.
3. Deduplication
   All objects that holds the exact same contnet are only stored once.

Files IPFS also defines a set of objects. The object model is similar to that of Git:

1. `block`: a varialbe-size block of data
2. `list`: a collection of blocks or other lists
3. `tree`: a collection of blocks, lists, or other trees
4. `commit`: a snapshot in the version history of a tree

4.2 Blockchain

This research utilizes the Bitcoin blockchain in order to record the proof of existence regarding the datasets. Before proceeding to explain how timestamping works, it would be useful to understand how the underlying Blockchain technology works.

Blockchain is a ledger organized in a distrubited manner, unlike traditional banking and payment systems. While traditional systems were managed under a centralized authority, blockchain attempts to mitigate abuse of such authority by dispersing the ledger entries while maintaining a consensus among participating nodes.

The most notable application is Bitcoin. Bitcoin has been attracting an incredible amount of attention among finance and technology industries (thus encouraging these two sectors to converge into a collaboration of a "Fintech" industry) as well as in the general public.

While this research is Blockchain agnostic, we will use the Bitcoin blockchain given its largest scale and available support among various public Blockchains.

4.2.1 Bitcoin Overview

According to its original anonymous inventor Satoshi Nakamoto, Bitcoin is "a purely peer-to-peer version of electronic cash"[24]. Nakamoto proposes to utilize digital signatures to give identities to transcantees and peer-to-peer network to resolve the double-spending problem.

4.2.2 Transactions

Transactions inform the allocation and relocation of Bitcoin according to its digital signatures. They exist like lines in a double-entry bookkeeping ledger, the entries being "inputs" and "outputs". These two act like debit and credit, the former leaving the account and the latter getting added to one. These inputs and outputs do not have to add up to the same amount. Inputs tend to be greater than outputs, while the difference implies a "transaction fee" for the miner including the transaction to the ledger.

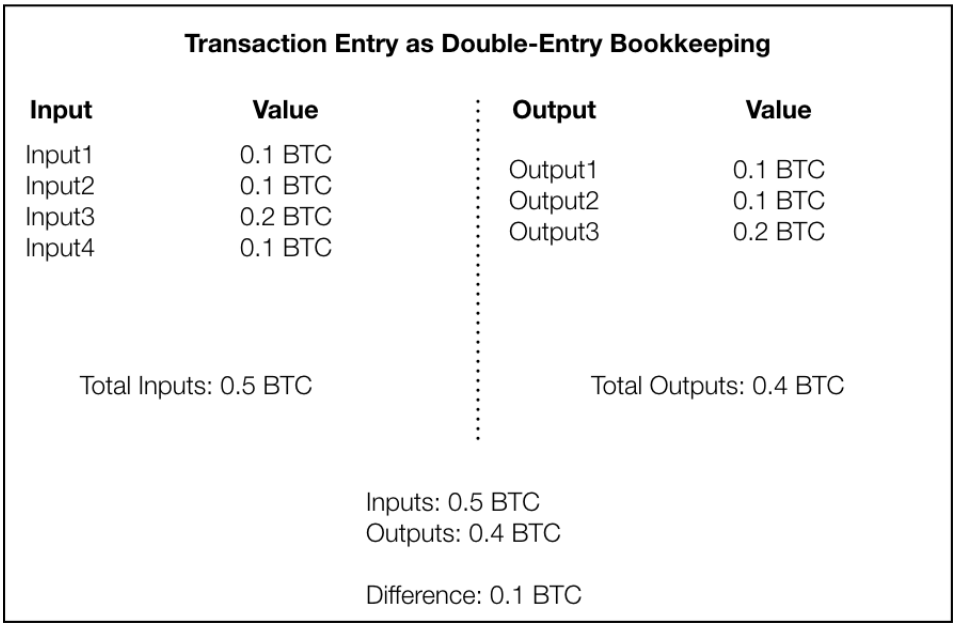

図 4.2: Transaction as double-entry bookkeeping

The transaction also contains proof of onwership for each amount of bitcoin in the form of a digital signature from the owner, which can be indepdently validated by anyone. So when an owner "spends" his bitcoins, he is signing a transaction that transfers value from a previous transaction over to a new owner identified by a bitcoin address.

A transaction has 4 stages within its lifecycle. They include:

1. Origitnation
2. Signature
3. Broadcasting
4. Mining

Transaction Structure

A transaction is a data structure that encodes a transfer of value from input to output. A transaction contraints a number of fields, as such in Table 4.1.

Transaction Outputs and Inputs

Bitcoin transactions are structured by units called *unspent transaction output*, or UTXO. UTXO are individible chunks of bitcoin locked to a specific owner, as recorded on the blockchain. Since they are indivisible, a spender must spend the entire UTXO in order to

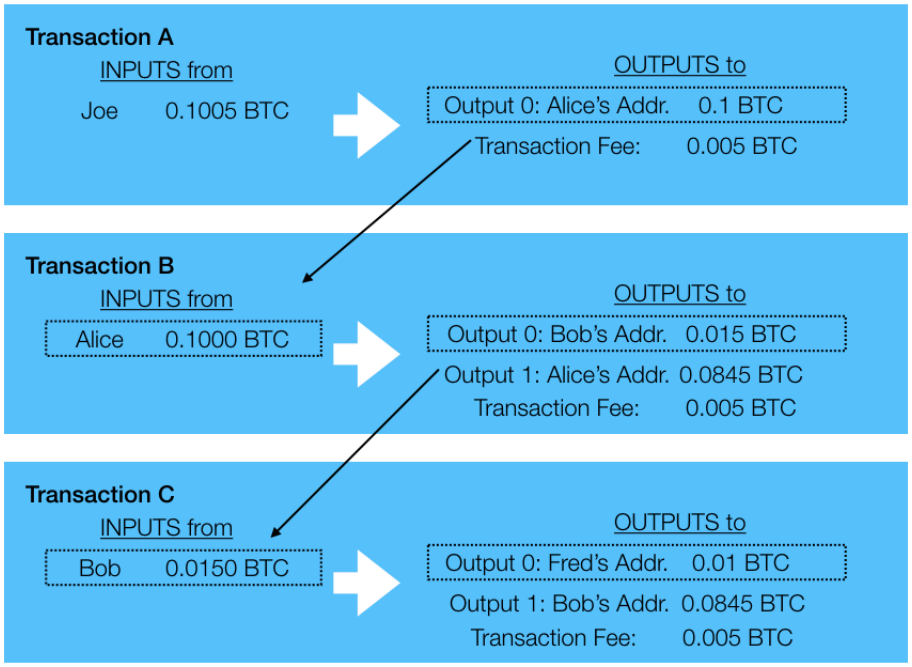

図 4.3: A chain of transactions, where the output of a previous transaction is used as an input of the next transaction

表 4.1: Structure of a Transaction

Size	Field	Description
4 bytes	Version	Specifies which rules this transaction follows
1-9 bytes (Variable)	Input Counter	How many inputs are included
Variable	Inputs	One or more transaction inputs
1-9 bytes (Variable)	Output Counter	How many outputs are included
Variable	Outputs	One or more transaction outputs
4 bytes	Locktime	A Unix timestamp or blocknumber

pay someone. For example, if Alice has 5 bitcoins wants to pay Bob 1 bitcoin, Alice can issue herself a transaction that consumes one UTXO of her posession as her input and produces two outputs: 1 bitcoin to Bob and 4 bitcoin to Alice. Therefore, the transaction generates change that will pay back the difference to the payer.

Transaction Outputs All bitcoin transaction creates outputs that are recorded on the bitcoin ledger. Once recognized by the bitcoin network, outputs will become available for

the owner to spend in a future transaction.

Transaction outputs contain two parts:

- An amount of bitcoins
- A *locking script* that "locks" this amount by specifying the conditions to be met to spend the output. Most scripts lock the output to a specific bitcoin address, namely the new owner

The structure of a transaction output is noted on Table 4.2.

表 4.2: Structure of a transaction output

Size	Field	Description
8 bytes	Amount	Bitcoin value in satoshis (10^{-8} bitcoin)
1-9 bytes (Variable)	Locking-Script Size	Locking-Script length in bytes
Variable	Locking-Script	Spending conditions defined by script

Transaction Inputs Simply put, transaction inputs point out to previously exhisting UTXO. They reference to specific UTXO by the transation hash and sequence number.

The structure of a transaction input is noted on Table 4.3.

表 4.3: Structure of a transaction input

Size	Field	Description
32 bytes	Transaction Hash	Pointer to transaction containing UTXO to spend
4 bytes	Output Index	Index number of UTXO to be spent, first being 0
1-9 bytes (Variable)	Unlocking-Script Size	Unlocking-Script length in bytes
Variable	Unlocking-Script	A script fulfilling conditions of UTXO locking script

Transaction Scripts

Bitcoin clients validate transactions by a script resembling Forth. Both locking and unlocking script are written in this scripting language. When a transaction is validated, the unlocking script in each input is processed with the corresponding locking script to check if it satifies the condition for spending. Most transactions are formed as Pay-to-Public-Key-Hash script.

A locking script, also known as *scriptPubKey*, "locks" the output and specifies the conditions for unlocking and spending in the future. It contains a digital signature produced from the private key. The unlocking script, also known as *sciptSig*, contains the digital signature.

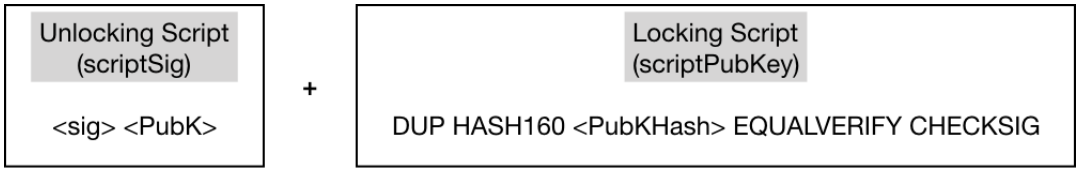

図 4.4: Combining scriptSig and scriptPubKey to evaluate a transaction script

When a transaction script is processed, the unlocking and locking scripts are concatenated. First, the unlocking script is read, then the locking script. If the scripts are executed without errors, the UTXO is made available to be spent.

Scripting Language

Script, the bitcoin transaction script language, is a stack-based execution language described in a Firht-like reverse-polish notation. It has these such attributes:

Turing Incompleteness While Script contains many operators, it is limited in way that disables loops or complex flow control. This renders the language not *Turing Complete*, so that scripts have limited complexity and predictable execution times. Such limitations ensure that the language can not be used to create an infinite loop or other forms of "logic bomb" that attacks the network.

Stateless Verification Script is stateless, in which there is no prior state when executing the script. This means that all information necessary to execute a script is self-contained within the script. This is beneficial as it makes the script verification predicatable across various nodes.

Standard Transactions

While the Script language can be written in many ways, the Bitcoin developers defined five "standard" transactions. Most nodes accept only these kinds of transactions. They include

- Pay-to-public-key-hash (P2PKH)
- Pay-to-public-key
- Multi-signature (up to 15 keys)
- Pay-to-script-hash (P2SH)
- Data output (OP_RETURN)

We will go over the most frequently used transaction type, Pay-to-Public-Key-Hash (P2PKH).

Pay-to-Public-Key-Hash (P2PKH) Most transactions processed on the bitcoin network are P2PKH. They have the locking script like this:

`OP_DUP OP_HASH160 <PubKey> OP_EQUAL OP_CHECKSIG`

The preceeding unlocking script looks like this:

`<Signature> <PubKey>`

The unlocking and locking scripts are combined, and when the script is evalated as TRUE, the user is authorized to spend the UTXO.

4.2.3 Blockchain Data Structure

The blockchain is often visualized as a "chain" of "blocks", so to speak. In this visualization, each block linked to the previous block, while each block contains a data structure that looks like "trees".

Each block has a hash as an identifier. It also references the hash of previous block, called *parent block*. This all leads back to the first block ever created, called the *genesis block*.

Block Structure

A block is a container structure that aggregates transactions to be included in the blockchain's public ledger. The structure is as listed on Table 4.4.

表 4.4: Structure of a block

Size	Field	Description
4 bytes	Block Size	in bytes
80 bytes	Block Header	Contains several fields
1-9 bytes (Variable)	Transaction Counter	Number of transactions
Variable	Transactions	Transactions in the block

Block Header

The block header contains three sets of block metadata. The first is the hash of the previous block; the second is a set of metadata relating to mining competition, such as *difficulty timestamp* and *nonce*; the third is the merkle tree root. The structure is listed on Table 4.5.

表 4.5: Stucture of Block Header

Size	Field	Description
4 bytes	Version	Version number to track protocol
32 bytes	Previous Block Hash	Refers to the hash of previous block in chain
32 bytes	Merle Root	Hash of root of merkle tree in this block's transactions
4 bytes	Timestamp	Approximate creation time of block (seconds from Unix epoch)
4 bytes	Difficulty Target	Proof-of-work difficulty target for this block
4 bytes	Nonce	Counter used for proof-of-work algorithm

Merkle Trees

A Merkle Tree is a data structure that efficiently summarizes and verifies the integrity of large sets of data. Also knowns as *binary hash tree*, each branch of the "tree" contains a cryptographic hash. The hash is computed from bottom up, eventually leading to a hash in the *merkle root* that summarizes the data in the tree. Assuming N nodes are hashed and summarized in a merkle tree, you can verify whether a node is included with at most $2 * log_2(N)$ calculations.

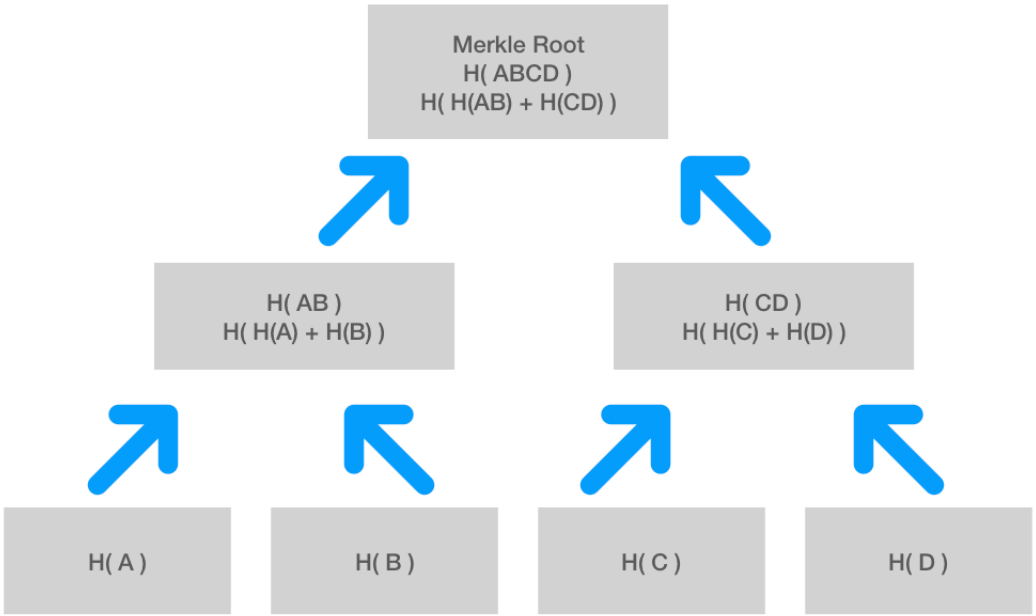

図 4.5: A merkle tree with each node hash calculated

4.3 OpenTimestamps

OpenTimestamps, developed by Peter Todd, is a timestamping tool that utilizes the Bitcoin blockchain.[25] In contrast to similar tools, OpenTimestamps provides three major advantages:

1. Trust
   OpenTimestamps utilzes the Bitcoin Blockchain, which is open and publicly-auditable, thus removing the need to trust authorities.

2. Cost
   OpenTimestamps scales indefinitely by combining multiple timestamps into a transaction, allowing this service to be provided for free.

3. Convenience
   OpenTimestamps can create stamps verifiable by thrid-parties within seconds, without the need to wait for a Bitcoin confirmation.

In the past, OpenTimestamps has been proved effective in timestamping large quantities of documents, such as the entirity of the Internet Archive in a single bitcoin transaction.[26] While alternatives exist, they are inefficient and costly given how they issuMe a stamp *per* document in lieu of aggregating with other documents; for example, poex.io[27] offers notorazation for 2.5 mBTC or about 35 USD using the conversion rate as of `January 11, 2017`[28].

4.3.1 Mechanism

OpenTimestamps works by using recoding hashes on the Bitcoin Blockchain. This section explains how it works.

Time Attestations

Each Bitcoin block header contains a field called "nTime". A Bitcoin block can be accepted by the Bitcoin network when the nTime is set approxiamtely to the time the block was created. While the time accuracy is not precise, it can be accurate for within two or three hours, and almost certainly within a day.

This makes Bitcoin useful as a *notary*, using Bitcoin blocks as *time attestations*.

Merkle Trees

The merkle root is contained within a Bitcoin block header. For example, the header for block 358,291 - exisitng on `May 28, 2015` - looks like this, with the merkle root in bold:

02000000b96394585a281b7e5f438fd1c9ed492645a1fd61cb380204000000000000000000**007ee445**
d23ad061af4a36b809501fab1ac4f2d7e7a739817dd0cbb7ec661b8a1e376755f58616186272def6

As discussed in 4.2.3, Merkle roots are calculated by taking all transactions within a block and creating a merkle tree.

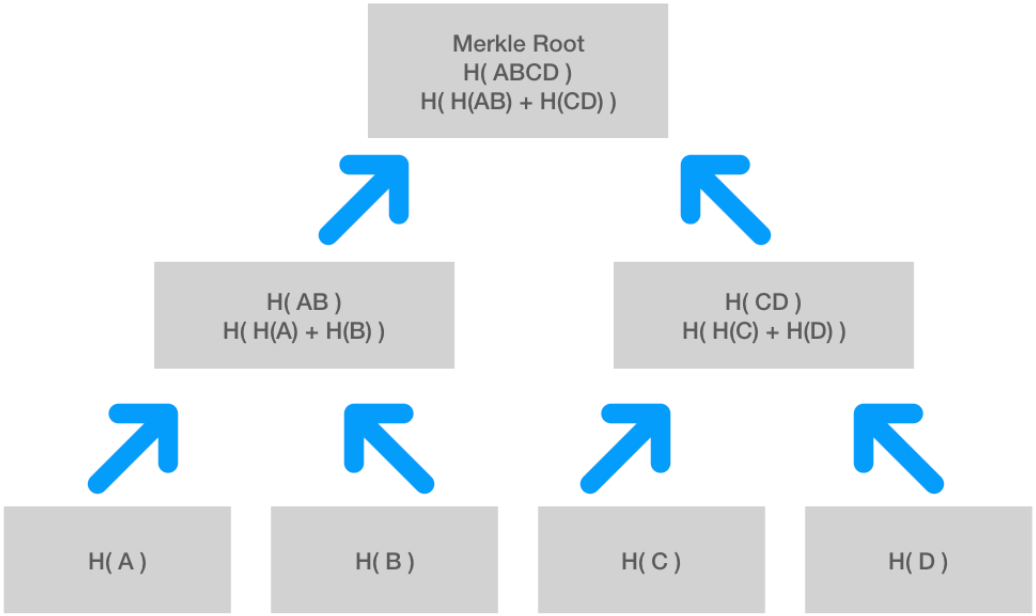

図 4.6: A merkle tree with each node hash calculated

Since the merkle root hash relies on all other hashes, should one hash underneath change, the the merkle root hash will change as well. For instance, if we change the last byte of the last transaction from block 358,291 from 0x00 to 0x01, then the merkle root changes completely:

```
Original:  007ee445d23ad061af4a36b809501fab1ac4f2d7e7a739817dd0cbb7ec661b8a
Modified:  87808e5a6a196e401bde8ebaf5b453825cdc3b66b87526bb355d519ae52ba42b
```

While it is possible to produce hash collisions among the nodes, but the possibility is said to be astronomically small.

Commitment Operations

Proofs are another way that the block header *commits* to the merkle tree; the merkle tree commits to the transactions, and the transaction are *committed* by the block header, because changing the transactions changes the header.

A timestamp proof in OpenTimestamps is a list of commitment operations applied to the message in sequence. For verification, one can replay the operations and confirm that the final result is a message already known to exist at a certain time. As every commitment operation is guaranteed to have a different result for a different input, it is difficult to change the message for timestamping without changing the result and invalidating the timestamp.

This is one example of opertaions in a timestamp:

```
$ ots info hello-world.txt.ots
File sha256 hash: 03
   ba204e50d126e4674c005e04d82e84c21366780af1f43bd54a37816b6ab340

Timestamp:
ripemd160
prepend 0100000001
   e482f9d32ecc3ba657b69d898010857b54457a90497982ff56f97c4ec58e6f980100000
append 88ac00000000
sha256
sha256
prepend
   a987f716c533913c314c78e35d35884cac943fa42cac49d2b2c69f4003f85f88

sha256
sha256
prepend
   dec55b3487e1e3f722a49b55a7783215862785f4a3acb392846019f71dc64a9d

sha256
sha256
prepend
   b2ca18f485e080478e025dab3d464b416c0e1ecb6629c9aefce8c8214d042432

sha256
sha256
append 11
   b0e90661196ff4b0813c3eda141bab5e91604837bdf7a0c9df37db0e3a1198

sha256
sha256
append
   c34bc1a4a1093ffd148c016b1e664742914e939efabe4d3d356515914b26d9e2

sha256
sha256
append
   c3e6e7c38c69f6af24c2be34ebac48257ede61ec0a21b9535e4443277be30646

sha256
sha256
prepend 0798
   bf8606e00024e5d5d54bf0c960f629dfb9dad69157455b6f2652c0e8de81
sha256
sha256
append 3
   f9ada6d60baa244006bb0aad51448ad2fafb9d4b6487a0999cff26b91f0f536

sha256
sha256
prepend
   c703019e959a8dd3faef7489bb328ba485574758e7091f01464eb65872c975c8

sha256
sha256
append
```

```
   cbfefff513ff84b915e3fed6f9d799676630f8364ea2a6c7557fad94a5b5d788

sha256
sha256
prepend 0
   be23709859913babd4460bbddf8ed213e7c8773a4b1face30f8acfdf093b705

sha256
sha256
verify BitcoinBlockHeaderAttestation(358391)
```

Scalability Through Aggregation

Unlike other blockchain timestamping systems that issues stamps for every document, OpenTimestamps is efficient because it aggregates document hashes into a single merkle tree that becomes recorded into a single transaction.

To improve upon this, OpenTimestamps provides a system of aggregation servers, where anyone can submit a digest to be timestamps. As of this writing, the two publicly available aggregation servers are `a.pool.opentimestamps.org` and `b.pool.opentimestamps.org`.

While these servers can become points of centralization, the influence is minimal. Should a server go offline, it will only become incoventiently inaccessible. The servers have no way to forge a fake timestamp, since the Bitcoin blockchain proves the validity of a timestamp.

第5章　Evaluation

In this chapter, we will cover the evaluation of the system designed in this research.

5.1　System Design

The core purpose of this sytem is to:

1. Make datasets as possibly available through distributed storage
2. Make the datasets verfiable with timestamps

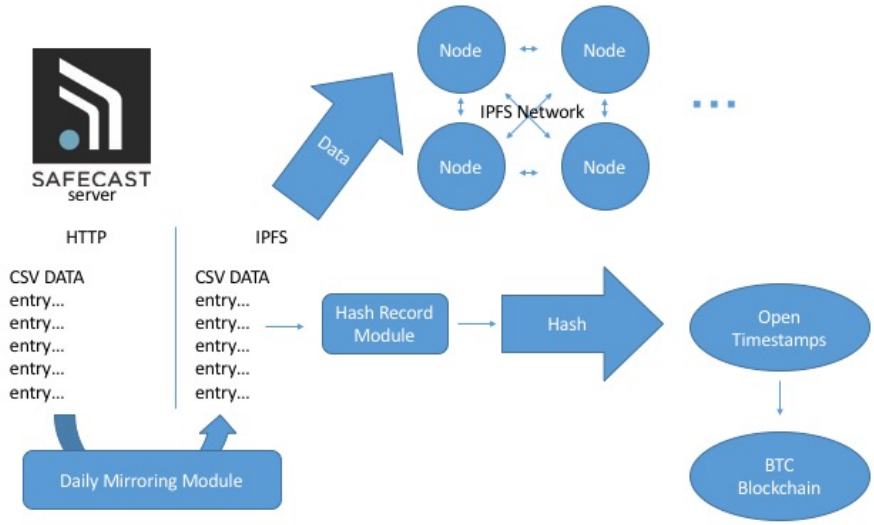

図 5.1: System Diagram

The system comproses of these components:

- SAFECAST server and API
  These servers are operated by SAECAST and provide access to their dataset through their API.
- IPFS Node
  An IPFS node will interact with the SAFECAST server.

- IPFS Network
  The IPFS network propogates data in an immutable manner.
- Open Timestamps (OTS)
  OTS creates verifiable timestamps using a blockchain.
- Bitcoin Blockchain
  The Bitcoin Blockchain will be used to maintain timestamps from OTS.

These components form a system which will operate as following:

1. A node establishes a connection with SAFECAST and IPFS.
2. The node fetches data in JSON from the SAFECAST database.
3. The node converts the data from JSON to csv.
4. The node sends the dataset in csv to the IPFS network.
5. The IPFS network propogates the dataset.
6. The node runs Open Timestamp, which first hashes the csv.
7. The node sends the hash to a timestamp calendar server, which produces a verifiable reciept.
8. After the data has been registered on the timestmap server, the node can check so using the reciept.
9. The calendar server aggregates other hashes from other submissions.
10. The calendar sever produces a merkle tree with submitted hashes.
11. The calendar server issues a transaction containing the merkle tree hash onto the Bitcoin blockchain.
12. After the data has been registered on the Bitcoin blockchain, the node can check so using the receipt. This will prove that the document was already in existence when it was registered on the blockchain.

The goals set above in 5.1 has been achieved in the following manners:

1. Datasets as available through seeding on IPFS
2. The existence of datasets on given dates are verfiable on the Bitcoin Blockchain using OpenTimestamps

5.2 Testing

In this section, we examined the efficacy of the constructed system by testing. The goal of this research is to make data more available through decenralized hosting. As a metric to measure availability, we chose to measure the length of time it takes for these data to become available online.

5.2.1 Methodology

For our benchmarking metric, we sough for the most pertinent factor in our distributed system. While immutability thorugh decentralization was a defining factor, the non-existence of a file in the future was not exactly testable in a limited timeframe. Therefore, we opted to benchmark the time it takes to have the mirrored dataset publicly available.

Testing Program

The program is designed to measure how soon it takes to have a newly uploaded file available online. The program operates as follows:

1. Generate File with Random String
   We create a new file that has probably never existed before by creating a text file containing psuedo-randomly generating gibberish sentences in the form of *lorem ipsum*.

2. Upload Generated File
   After the file has been created, we upload the file onto IPFS. When the upload is finished, the `ipfsapi` module returns the allocated IPFS hash.

3. Start Timer
   We start the timer for the benchmark as we begin to resolve for the newly generated textfile just uploaded onto IPFS.

4. Resolve for File on IPFS
   We run `ipfs resolve` on the hash of the newly generated textfile.

5. Stop Timer
   We stop the timer when the resolving is complete.

6. Write Result
   The result is written in a different textfile as numerical values representing the seconds to complete the resolution.

To gather more data, we loop the above process several times. In this experiement, the benchmark has been run 100 times.

5.2.2 Configuration

In this section we describe our hardware and software configuration in which operation and testing was conducted.

Hardware

We used a VPS serviced by ConoHa, a subsidiary of GMO Internet group.[29] The VPS operates using the OpenStack platform.[30] The hardware configuration under the VPS is listed on Table 5.1.

表 5.1: Hardware Configuration

Component	Spec	Notes
CPU	Intel Xeon @ 2.60GHz	Cores: 2
Memory	1GiB	
Memory	50GB	

Software

For our system, we chose to use the Linux distrobution of Debian. Software was installed accordingly as noted on Appendix A.

表 5.2: Software Configuration

Software	Version
Linux Debian	4.9.30-2+deb9u2
Python	2.7.13
Python3	3.5.3
Bitcoin Core	0.15.1
IPFS	0.4.13

5.2.3 Results

The result is graphed as a boxplot on Graph 5.2.
The statistics of the sampled data is listed below:

```
n = 100

MIN: 0.09340906
```

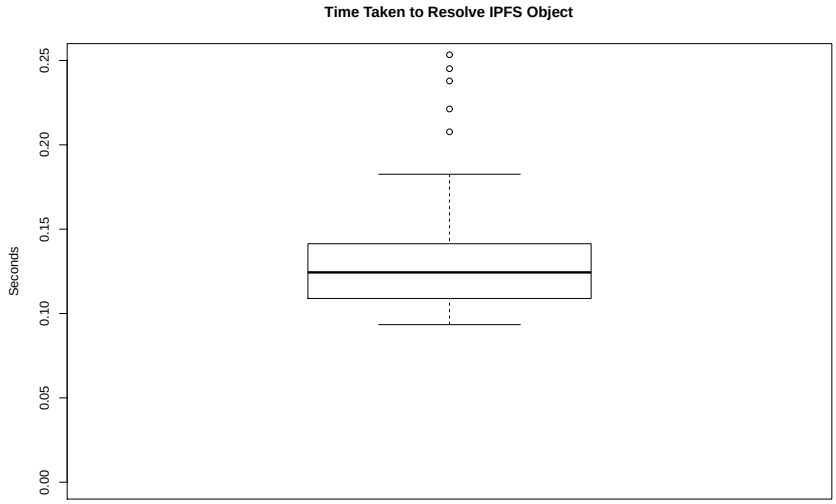

図 5.2: Evaluation Result

```
MAX: 1.459512
AVE: 0.1486072
1st Quartile: 0.10895246
Median: 0.12437451
3rd Quartile: 0.14129418
```

Given that 75% of the measurements are less than 0.14129418 seconds, we can say that the system performs effectively to a practically useful extent.

第6章　Conclusion

This research proposed a method to tackle the risk of abused trust in centralized hosting of Open Data by distributing the hosting through IPFS and ensuring the integrity with the Bitcoin Blockchain and Open Timestamps. In Open Data, it is important to ensure the integrity of the data with as little trust as possible. To do so, we moved the hosting of the data from a cetnralized model to a decentralized alternative. In our experiment, we showed that IPFS can serve data with just as much utility as centralized methods, allowing data to be accessible almost as soon as it is uploaded.

付 録A Setting Up the Scraping System

A.1 Dependencies

- Python
- Pandas
- IPFS
- ipfsapi
- Bitcoind
- Open Time Stamps (OTS)

A.1.1 Python

The program for this project was implemented in Python. While the main script is written to work under Python version 2.7.13, external libraries dependent on Python3, namely Open Time Stamps, operated under version 3.5.3. As for the package manager `pip` [31], we used version 9.0.1.

A.1.2 Pandas

Summary

Pandas[32] is a data analysis library for python. It is useful for manipulating numerical data.

Usage

Pandas is utilized to convert JSON data retrieved from the SAFECAST API into csv.

Installation

Users can install pandas module using package managers like pip.

```
pip install pandas
```

A.1.3 IPFS

Summary

IPFS[33], an acronym for "InterPlanetary File System", is a distributed file system enabled by P2P technologies. There are client API liraries implemented in Javascript[34] and Python[35] provided by the developers. We use the latter.

Usage

`ipfsapi` helps us upload files onto IPFS via Python interface.

Installation

First, we need to install IPFS to handle the communication within the P2P network.

1. Download IPFS from ipfs.io[36]
2. Extract the package by running this on commandline: `tar xvfz go-ipfs.tar.gz`
3. To place the binary in the appropirate directories, run: `./go-ipfs/install.sh`

This should install the IPFS programs into the appropriate directories.
As for the python module, users can install ipfsapi module using package managers like pip.

```
pip install ipfsapi
```

A.1.4 Open Time Stamps(OTS)

Summary

Open Time Stamps[2] proves the existance of data in a certain time by recording its hash on the Bitcoin blockchain. Unlike other timestamping systems, OTS aims to be efficient by collecting hashes from various documents into a single merkle tree and publishing the root merkle hash onto the Bitcoin blockchain.

Usage

We use `ots` program to submit the dataset hash onto the calendar server provided by Opentimestamps, and also to verify the integrity of these datasets.

Installation

Users can install ots module using package managers like pip.

```
pip install ots
```

A.1.5 Bitcoind

Summary

`bitcoind` is the daemon that runs the node to participate in the bitcoin system. It can be in many regards with the bitcoin, such as creating, parsing, modifying transactions, examining the entries of the Bitcoin blockchain, and such. While many other nodes are available on the Internet, it is useful to prepare a local node to ensure the integrity of data the user deals with.

Usage

We use `bitcoind` to verify the integrity of the hashes recorded by OTS.

Installation

We assume that the destination system is Linux, as `bitcoind` is a server software intended to be running continuously on a server like Linux rather than a personal computer.

1. Download Bitcoin Core from bitcoin.org[37] and verify that you have a secure connection.
2. Extract the file with
   `tar xzf bitcoin-0.ab.c-x86_64-linux-gnu.tar.gz`
   (Replace `a, b, c` with downloaded version number.)
3. Install the binaries locally with
   `sudo install -m 0755 -o root -g root -t /usr/local/bin bitcoin-0.14.2/bin/*`
4. Start bitcoind with
   `bitcoind -daemon`

Further installation intruction can be found on bitcoin.org[38]

A.2 Operation

1. Bitcoind
2. Install Script
3. Set up crontab

A.2.1 Pruning in Bitcoind (Optional)

When you first run `bitcoind`, the program will start connecting to peers and download the entire Bitcoin blockchain. This can take up a significant amount of storage space; as of Jan 4th 2017, the total size is 150GB.[39] While storing the entire blockchain is a requirement for using `bitcoind` for mining such as validating transactions and broadcasting blocks to other nodes, it is not necessary for our purpose of verfying our hashes in the blockchain. Therefore, we can keep the size of the blockchain stored locally to the minimum required size by pruning.

In the bitcoin system, there are four types of data: the raw blocks as received over the network, the undo data, the block index and the UTXO set (the last two being databases). The databases are built from raw data. Block pruning allows `Bitcoind` to delete raw block and undo data once it's been validated and used to build the databases. After that, raw data is only used to relay blocks to other nodes, to handle reorganizations, to look up old transactions, or for rescanning the wallet. The block index continues to hold metadata regarding all blocks in the blockchain.[40]

To enable pruning, you can edit `bitcoin.conf` and write in `prune=<N>`, where N is the number of MiB to allot for raw block and undo data. The minimum allowed is 550MB.

A.2.2 Install Script

To install the script made for this thesis, you can download it from a Git repository.[41] After downloading, make a new directory for the SAFECAST dataset and move the program there. You can run the program from the directory by entering the following into the commandline:

```
python scraping.py
```

A.2.3 Set up crontab

To run the above program periodically, we can set up a scheduling system with `cron`.

The schedule for cron is kept in a crontab, or a cron table file. The cron table is organized into the following:

```
# ┌──────────────────────────── minute (0 - 59)
# │ ┌────────────────────────── hour (0 - 23)
# │ │ ┌──────────────────────── day of month (1 - 31)
# │ │ │ ┌────────────────────── month (1 - 12)
# │ │ │ │ ┌──────────────────── day of week (0 - 6)
# │ │ │ │ │
# │ │ │ │ │
# * * * * *       command to execute
```

To schedule the program, do the following:

1. Open `crontab -e` on the commandline.
2. On the left portion of the `crontab`, enter a timedate you desire. For example, you can schedule the program to run on 12:00 AM everyday by setting `0 0 * * *` . Do not forget to seperate each number with spaces.
3. On the right portion of the `crontab`, enter a command to change to the directory containing the program and run the program. It would look like this:
   `cd /path/to/directory/ && python scraping.py`

A.3 Verification

Before a file can be verified for its existence in a given time, we need to wait until the file hash has been registered on the blockchain. This can take about 24 hours. Until then, we can only see that the file has been submitted to the calendar server.

A.3.1 Online Verification

If you don't want to install `bitcoind` and download the blockchain, you can still verify the file online via sites like openstamps.org[2]. However, there is a risk that the website may be modified to act maliciously. Therefore this method is not reccommended for secuirty reasons, but is provided for expediency.

To verify online, open the file and the `.ots` receipt in the browser.

A.3.2 Local Verification

To verify a hash locally, do the following:

1. Change to the directory containing the datasets.

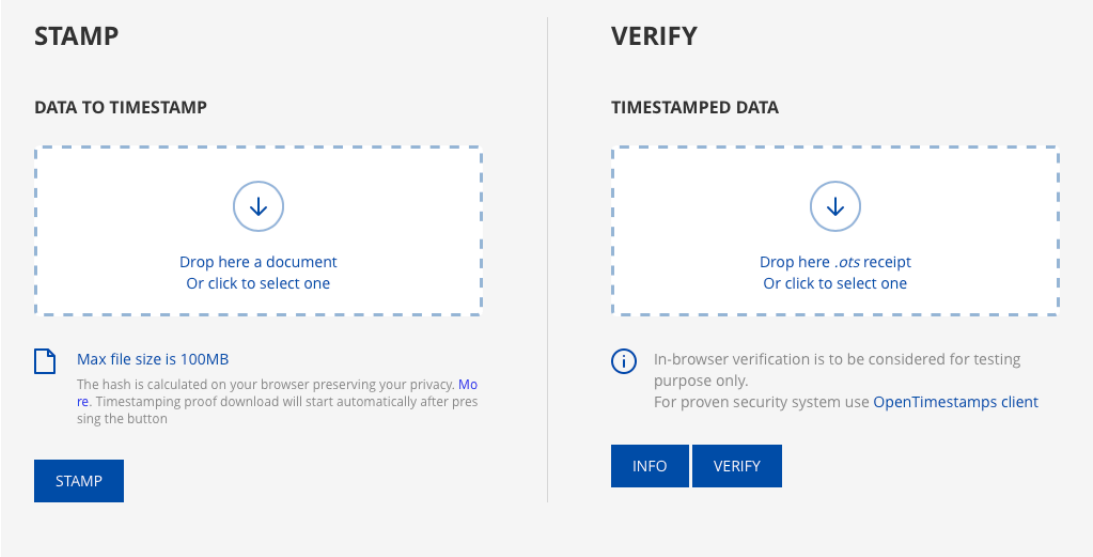

図 A.1: Online verification via opentimestamps.org[2]

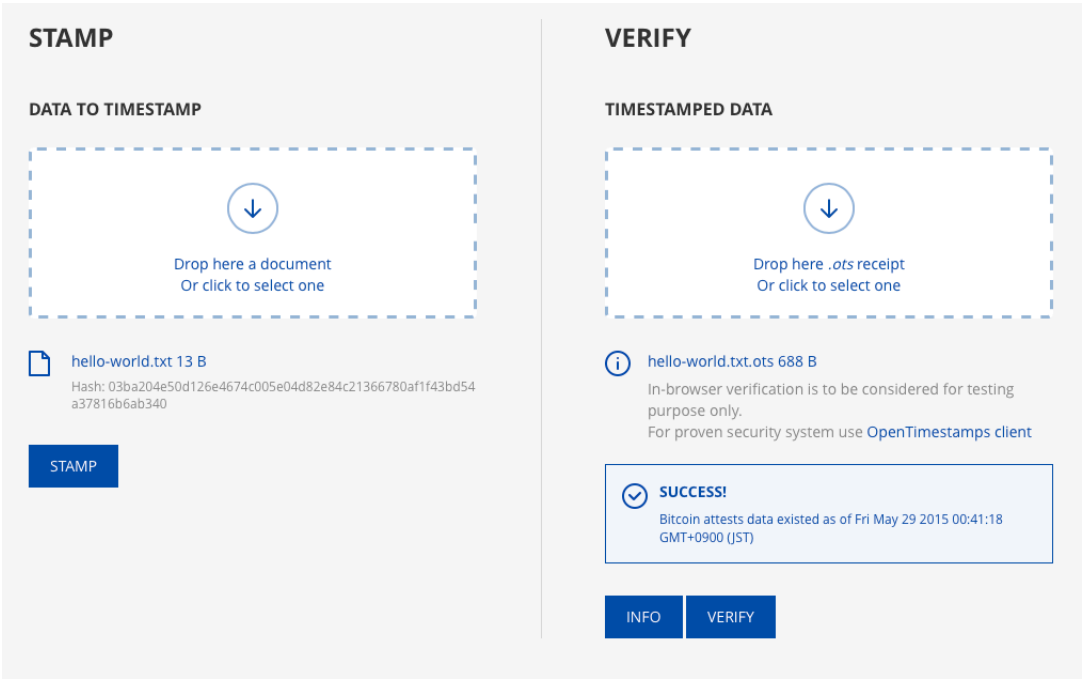

図 A.2: A successful online verification

2. type in the following in the commandline:
   `ots verify filename.csv.ots`
   (Make sure the filenames of both .csv and .ots matches.)

An output of a successful verification looks like this:

```
ots verify hello-world.txt.ots
Assuming target filename is 'hello-world.txt'
Success! Bitcoin attests data existed as of Fri May 29 00:41:18 2015 JST
```

A.4 Seeding Only

You can also help with the project by making the dataset more available. To contribute, you can copy the contents of the IPFS directory and host it from your node. To do so, you can run the following command:

```
ipfs pin add QmY54zq6q81jn9fep23REJPA9UtbKrfH4bLALnCSnk7R5a
```

(You can change the IPFS path (the last parameter) into another path accordingly.)

付 録B　Submitting data to SAFECAST

Contributing data to SAFECAST on their website consists of 6 steps.

1. Upload
2. Processing
3. Adding Metadata
4. Submission
5. Approval
6. Live

1. Upload

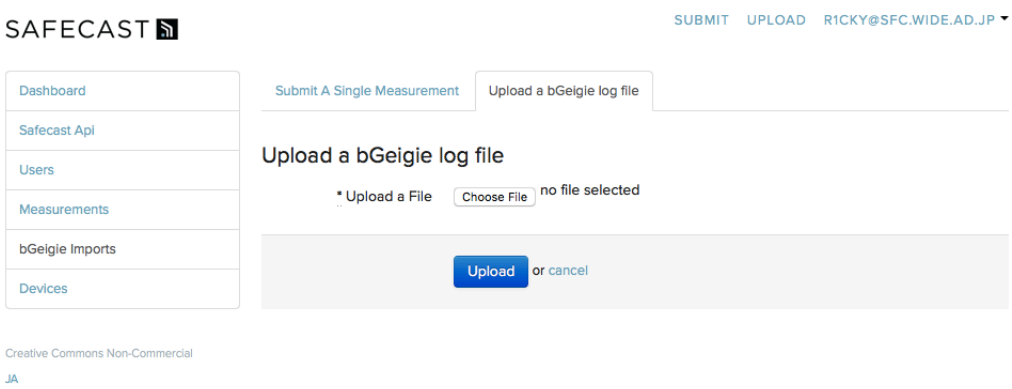

図 B.1: Uploading bgeigie data on SAFECAST

First, the user takes out the microSD card previously inserted in their bGeigie counters and plugs it into their computer. The user then accesses the website[9] and uploads the log file they wish to contribute. This feature can be accessed via a link labeled “UPLOAD”.

After uploading the log file, the dashboard indicate that the file has been taken into the SAFECAST server. At this point, the data has not been processed yet. Also, the user can edit the metadata if he desires to.

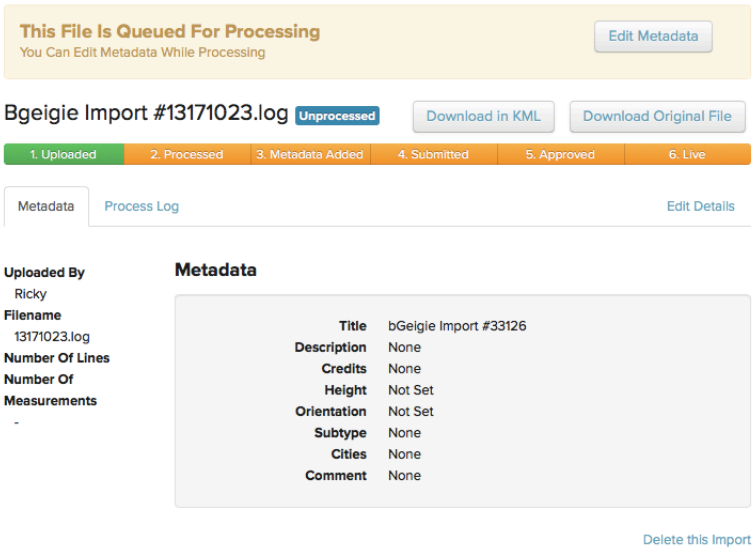

図 B.2: Screen after data has been uploaded

2. Processing

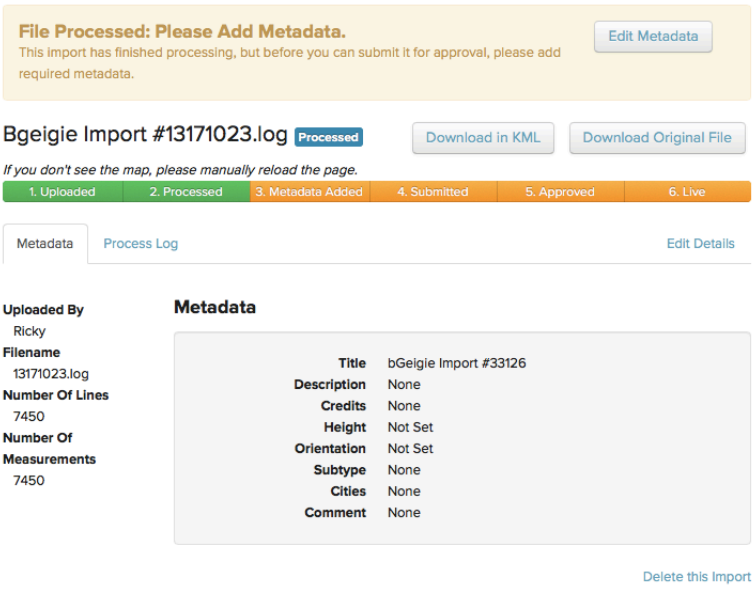

図 B.3: Screen after data has been processed

After a few moments, the user can reload the website and see that the data has been processed. However, it is still necessary to add metadata.

3. Adding Metadata

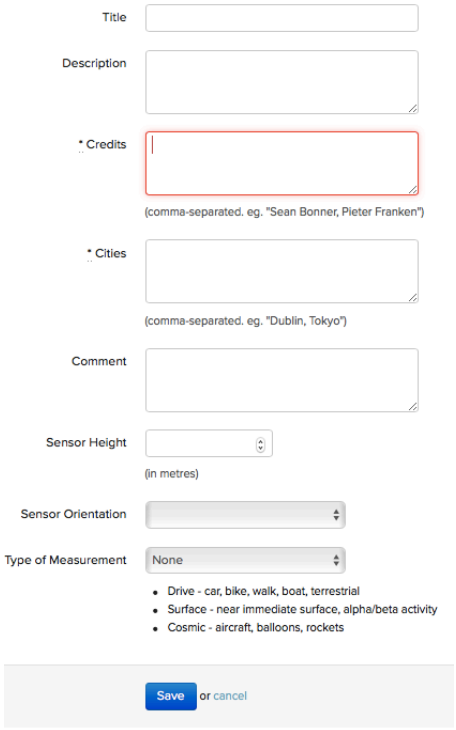

図 B.4: Prompt to enter metadata

When the user clicks to add metadata, a prompt appears with input fields such as: title, description, credits, comment, sensor height (in meters), sensor orientation, and type of measurement such as Drive (car, bike, walk, boat, terrestrial), Surface (near immediate surface, alpha/beta activity), and Cosmic (aircraft, balloons, rockets). Out of these, only credits and cities are required. Such entries are used to enhance the details of the entries and improve understanding of the conditions in which the measurements were taken.

4. Submission

Once the metadata is entered, the log can be submitted for approval. However, the user can also choose to reject the submission.

4. Submission

After submitting, it can take about a day for SAFECAST volunteers to approve the log submission and accept it as part of its dataset.

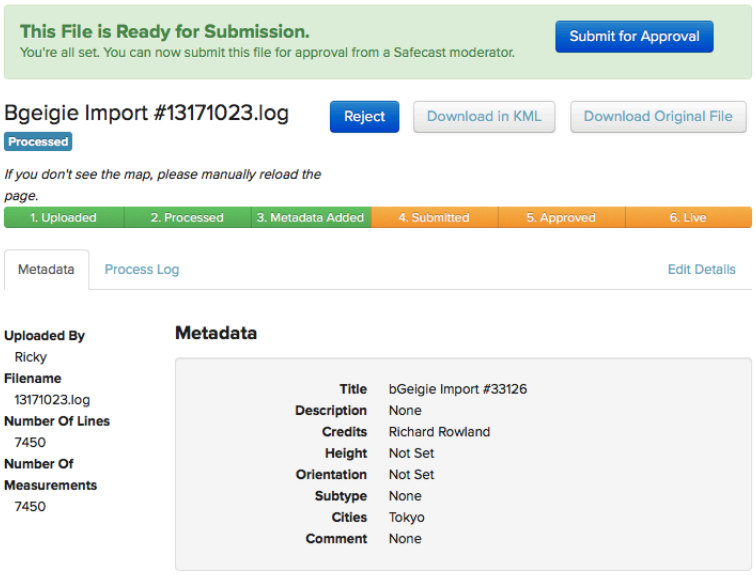

図 B.5: Screen indicates that log is ready to be submitted

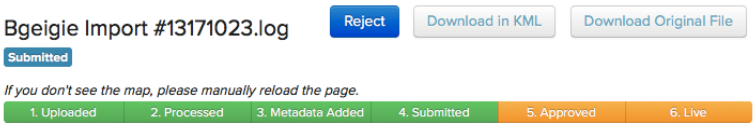

図 B.6: Screen indicating that log has been submitted for approval

5. Approval

When the log submission get approved, you will get an email notification. This usually happens within 24 hours. The email should look like this:

> Your Safecast import has been approved - 13171023.log
>
> Your Safecast import has been approved. Click here to view it.

You can click on the link to see your data online.

6. Live

Once you open the link to your submitted data, you can confirm that it's live.

図 B.7: Screen indicating that log is live

Bibliography

[1] Andreas M. Antonopolous. Mastering Bitcoin: Unlocking Digital Cryptocurrencies.

[2] OpenTimestamps. OpenTimestamps. `https://opentimestamps.org`.

[3] The Open Definition - Open Definition - Defining Open in Open Data, Open Content and Open Knowledge.

[4] Ray P Norris. HOW TO MAKE THE DREAM COME TRUE: THE ASTRONOMERS' DATA MANIFESTO.

[5] TimothyH. Vines, ArianneY.K. Albert, RoseL. Andrew, Florence Débarre, DanG. Bock, MichelleT. Franklin, KimberlyJ. Gilbert, Jean-Sébastien Moore, Sébastien Renaut, and DianaJ. Rennison. The Availability of Research Data Declines Rapidly with Article Age. *Current Biology*, 24(1):94–97, jan 2014.

[6] Faculty of 1000 Ltd. *F1000Prime.*

[7] About Safecast — Safecast.

[8] Kate Dougherty. Parts List, 2017.

[9] SAFECAST. The safecast api. `https://api.safecast.org`.

[10] Trump's EPA has started to scrub climate change data from its website - LA Times.

[11] HP. There is no free lunch with distributed data white paper Consistency, availability, and partition-tolerance trade-offs on distributed data access systems, 2005.

[12] Mikito Takada. Distributed systems for fun and profit. `http://book.mixu.net/distsys/index.html`.

[13] Mike Burrows. The Chubby Lock Service for Loosely-Coupled Distributed Systems.

[14] Doozer. Doozer. `https://github.com/ha/doozerd`.

[15] Apache Hadoop. Zookeeper. `https://zookeeper.apache.org/doc/r3.3.3/zookeeperStarted.html`.

[16] HashiCorp. Consul. `https://www.consul.io`.

[17] CoreOS. etcd. https://coreos.com/etcd/docs/latest/.

[18] Apache. Cassandra. http://cassandra.apache.org.

[19] Basho. Riak. http://basho.com/products/.

[20] MongoDB Inc. MongoDB. https://www.mongodb.com.

[21] Apache. CouchDB. http://couchdb.apache.org.

[22] Juan Benet. PFS - Content Addressed, Versioned, P2P File System (DRAFT 3). https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

[23] OpenDHT. What are Distributed Hash Tables ? https://github.com/savoirfairelinux/opendht/wiki/What-are-Distributed-Hash-Tables-%3F.

[24] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf.

[25] Peter Todd. OpenTimestamps: Scalable, Trustless, Distributed Timestamping with Bitcoin. https://petertodd.org/2016/opentimestamps-announcement.

[26] Peter Todd. How OpenTimestamps 'Carbon Dated' (almost) The Entire Internet With One Bitcoin Transaction. https://petertodd.org/2017/carbon-dating-the-internet-archive-with-opentimestamps.

[27] PoEx Co. Ltd. Proof of Existence. https://poex.io.

[28] You Me BTC. Convert BTC, mBTC, Bits, Satoshis, USD, EUR, and More. https://youmeandbtc.com/bitcoin-converter/convert-btc-mbtc-bits-satoshis-usd/.

[29] OpenDHT. ConoHa by GMO. https://www.conoha.jp.

[30] openstack. openstack. https://www.openstack.org.

[31] pip project. pip. https://pip.pypa.io.

[32] The pandas project. Python Data Analysis Library — pandas: Python Data Analysis Library. https://pandas.pydata.org.

[33] Protocol Labs. IPFS is the Distributed Web. https://ipfs.io.

[34] IPFS. IPFS HTTP API in JavaScript. https://github.com/ipfs/js-ipfs-api.

[35] IPFS. IPFS HTTP API in Python. https://github.com/ipfs/py-ipfs-api.

[36] IPFS. go-ipfs. https://dist.ipfs.io/#go-ipfs.

[37] Bitcoin Core. Download Bitcoin Core. https://bitcoin.org/en/download.

[38] Bitcoin Core. Running A Full Node. https://bitcoin.org/en/full-node.

[39] Blockchain.info. Blockchain Size. https://blockchain.info/charts/blocks-size.

[40] Bitcoin Core. Block File Pruning. https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning.

[41] Richard Rowland. Scraping.py. https://github.com/rg-kumo/r1cky_theis/blob/master/program/scraping.py.