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 環境情報学部
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 Enviornment 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 managed under the belief that data itself should be openly used, and should therefore be promoted without patents, copyrights, or such enforcements of intellectual properties. Open data can be defined as such:

> Open Data is data that can be freely used, re-used, and re-distributed, and the most restrictive condition would be the requirement of attribution and inheritance of license conditions.

Openness is determined under these criteria:

- Availability and Accessibility
  Dataset is available at a reasonable cost. For instance, downloadable on the Internet. Data itself should be provided in a format that is convenient and editable
- Reuse and Redistribution
  Dataset is adaptable to various use cases, such as mergers with other datasets, by allowing reuse and redistribution.
- Universal Participation
  Dataset can be used, reused, and redistributed by anyone without discriminating against certain individuals or organizations. For example, “non-commercial” licenses that exclude “commercial” use are not welcomed.

The greatest reason to promote open data is interoperability. Interoperability is the ability for various organizations and systems to collaboration across organizational borders. Interoperability promotes development of more advanced systems by combining modules together. With the ability to mix datasets, it advances the development of knowledge, insight, and products, allowing innovation to prosper.

Open Data plays an important part in scientific research. For example, better access increase the rate of scientific discoveries.[2] 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.[3][4]

2.1.1 SAFECAST

As an example of the use of Open Data, SAFECAST provides radiation data sampled from their network of sensors.[5] 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.[6] 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.[7] 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.[8]

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[9] postulates the difficultly of hosting a decentralized database in its acronymn[10]:

- 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.

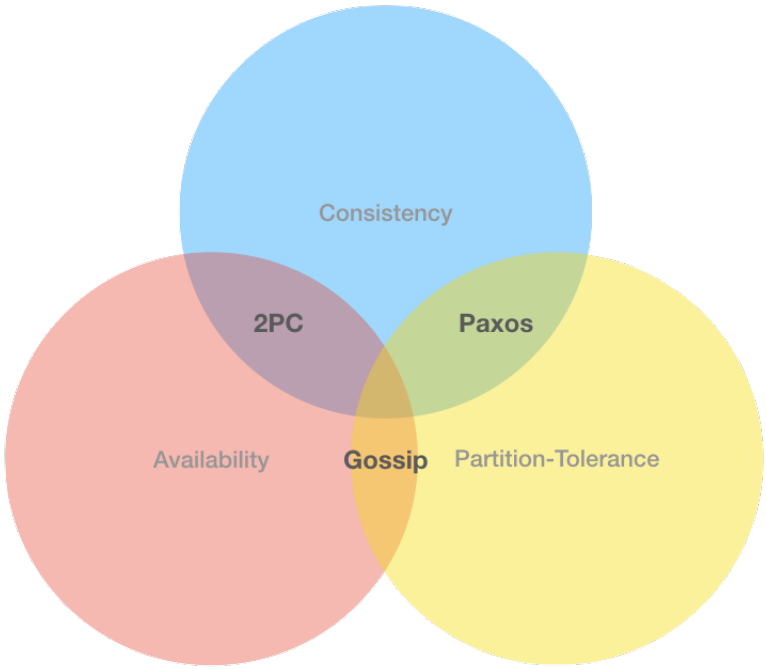

図 3.1: Design choices based on the CAP theorem

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[11], Doozer[12] (Paxos)
- ZooKeeper[13] (Zab)
- Consul[14], etcd[15] (Raft)
- N/A (Viewstamped Replication)

AP

- Cassandra[16]
- Riak[17]
- Mongo[18]
- Couch[19]

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.

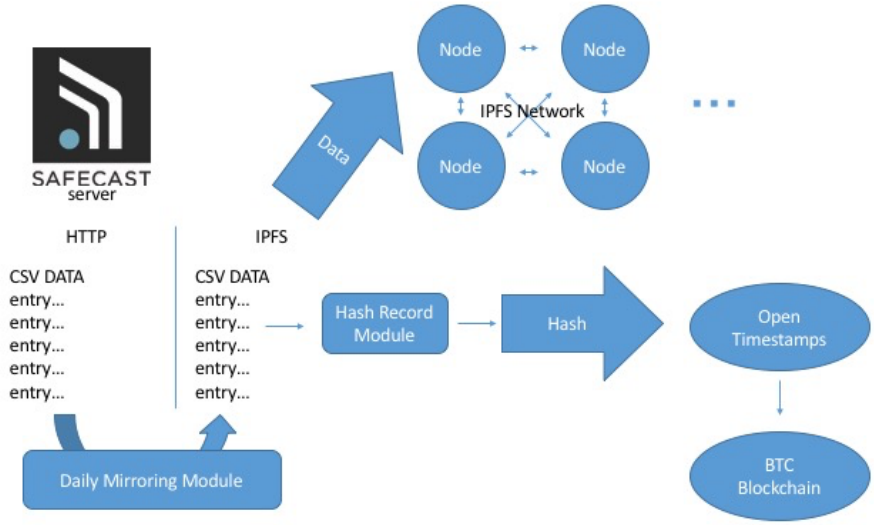

図 4.1: System Diagram

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

IPFS, an acronym taken from InterPlanetary File System, is a distributed file system that utilizes Peer-to-peer (P2P) technologies connecting network participants directly.[20] In comparison to the already-popular P2P technologies like BitTorrent, rather than connecting peers on a per file basis, IPFS connects participants to a single big file directory and makes it accessible like the web. In addition, the directory itself is designed after file tracking software like Git. IPFS combines Distributed Hash Tables, Block Exchange, and Self-Certifying Namespaces. This architecture helps prevent failures often caused by single points of failures.

4.1.1 Background

This section will explain the technologies used to realize IPFS.

DHT Distributed Hash Tables are often used to exchange metadata within P2P networks. It works like a dictionary service from which one can extract data entries corresponding to a key. In DHT, each participating node is given a randomly-generated unique key. Values stored on a network are also identified by a randomly-generated unique key. A participating node accumulates information on other nodes and their IP addresses and generates a routing table that represents the network partially from its perspective. The routing table stores node IDs in a binary tree. Also, each node stores data. For instance, to find a node that stores data corresponding to the key `H`, the searching node contacts the nodes within its routing table and seeks the node closest to `H`. The query contains the key, while the response contains whether the node stores data corresponding to `H` and a pair of Node ID and IP address from a node nearest to `H` according to the responder ' s routing table.

Popular DHT implementations include `Coral`, `Kademlia`, and its retuned variant `S/Kademlia`.

Block Exchange BitTorrent, one of the most prevalent P2P technologies, served as inspiration for the design of Block Exchange. BitTorrent enabled untrusting peers to connect and share files successfully. IPFS borrows the following perspectives from BitTorrent:

1. BitTorrent uses a "Tit-for-tat" strategy to reward nodes cooperative to sharing files and stymie nodes that don ' t.
2. BitTorrent prioritizes the rarest file pieces first.
3. PropShare improves upon the standard "tit-for-tat" bandwidth allocation strategy employed in BitTorrent.

File Change Tracking Software that track file changes are well-suited to track and distribute differences of each file. A popular software like Git improves upon a data structure called Merkle Trees and makes distribution more efficient. Merkle Trees are used in such a manner as:

1. The isomorphism between components of a file system and Merkle Trees are as such:
   - Immutable objects - `blob`
   - Directories - `tree`
   - Changes - `commit`
2. Objects are cryptographically converted to and represented by hash strings
3. Links to other objects are embedded to form a Merkle Tree
4. Metadata regarding file changes are used as reference points
5. Processes following file changes are limited to updates of reference points and adding of objects
6. Distributing file changes can be done by moving objects and updating remote reference points

Self-Certified Filesystems SFS, an acronym of Self-Certifying File System, is (a) a trusted distributed chain and (b) a means to promote an egalitarian global namespace. SFS can be expressed in such syntax:

```
/sfs/<Place>:<Host ID>
```

The `place` is a server ' s network address, while the `Host ID` is a hash produced from the concatenation of the public key identifying the server and the network address. This makes the name of the SFS File System certify the server. This lets users authenticate servers by its public key, share secrets, and protect connections afterwards. All SFS existences share a global namespace, cryptographically allocating names without using centralized authorities.

4.1.2 Design

IPFS is modeled after the aforementioned DHT, BitTorrent, SFS, and such P2P technologies in existence. This enables IPFS to provide the following features:

Node Independence Each node is allocated a `Node ID` based on the hash of its public key. A node has its public key and the secret key derived from.

Network The IPFS network contain nodes frequently communicating to each other. The network contains the following features:

- Transport
  IPFS can use any transport protocol
- Reliability
  IPFS can guarantee reliability should the underlying protocol lack such features
- Networking
  IPFS employs Network Address Translation schemes from Interactive Connection Establishing methods
- Integrity
  Data integrity is confirmed by its accompanying hash
- Authenticity
  Authenticity of sent data is confirmed by the sender ' s public key

Routing The routing system in an IPFS node can find (a) network addresses of other nodes and (b) routes to distribute given objects. This is realized by DHT. Values less than 1KB are stored directly in the DHT, while greater values are stored as references to nodes that distribute them.

Block Exchange Data exchange in IPFS is realized by BitSwap, a protocol inspired by BitTorrent. In BitSwap, nodes have a `want_list` and `have_list` of blocks sought and stored respectively. As a feature differentiating from BitTorrent, exchangeable blocks are not limited to one torrent directory, but encompass any file formats.

Merkle Tree DHT and BitSwap enables IPFS to construct large-scale P2P systems that swiftly and robustly distribute blocks. On top of that, IPFS constructs Merkle trees that link objects by their cryptographic hashes. Features of Merkle trees include:

1. Content-addressing
   All content are identified by their hashes
2. Tamper-proof
   All content integrity is insured by their hashes
3. Deduplication
   Objects with the same hashes are stored only once

Files IPFS identifies objects in the following categories:

1. `block`: Size-variable blob of data
2. `list`: Collection of blocks or lists
3. `tree`: Collection of block, list or trees
4. `commit`: A state of change within 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

Bitcoin, coined by its pseudonymous inventor Satoshi Nakamoto, developed with an aim to be a completely P2P electronic cash system.[21] Nakamoto utilized electronic signatures to give individual identities to transactees and solved the double-payment problem using a P2P network.

4.2.2 Transactions

Transactions determine the distribution and redistribution of Bitcoins based on digital signatures. It exist like a double-netry ledger containing columns of "input" and "output". These two act like liabilities and asset, the former leaving the account and the latter entering. They don ' t necessarily have to be the same amount. In fact, it ' s common that the input is greater than the output, with the difference being the "transaction fee".

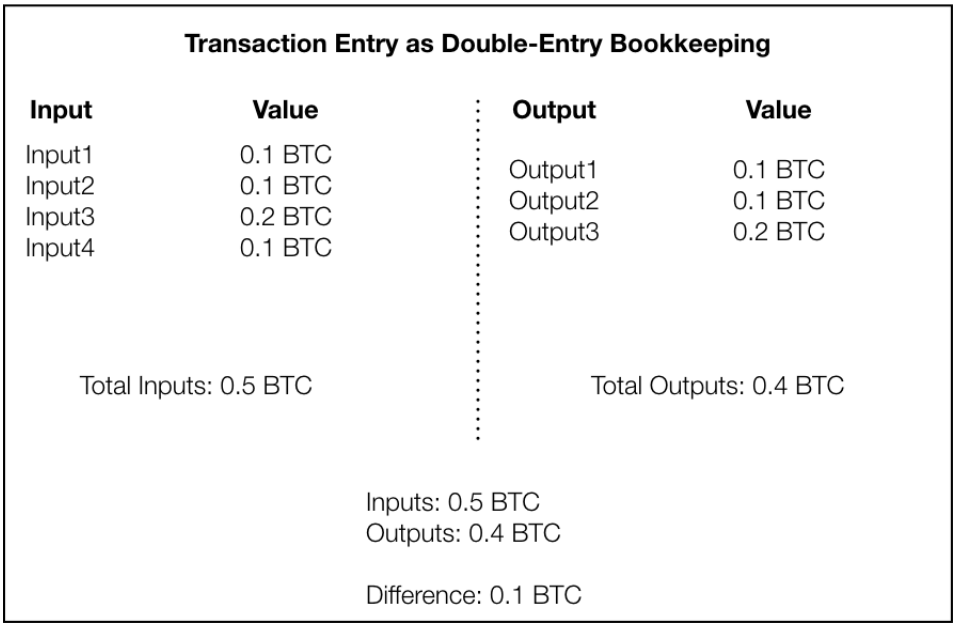

図 4.2: Transaction as double-entry bookkeeping

The transactions label Bitcoin ownership verifiable via digital signatures. This is verifiable by anyone. Therefore, when a Bitcoin owner "pays", that person is transferring value gained from a previous transaction to another bitcoin address while authorizing with his digital signature.

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

1. Generation
2. Signing
3. Broadcasting
4. Mining

Transaction Structure

A transaction is a data structure that indicates the transfer of value with inputs and outputs. Transaction contains several entries, as labeled on Table 4.1.

Transaction Outputs and Inputs

Bitcoin transactions are constituted by units called UTXO, which stands for Unspent Transaction Outputs. UTXO is an indivisible amount of Bitcoins allocated to a given owner. Since it is indivisible, the payee has to consume the entire UTXO. For example,

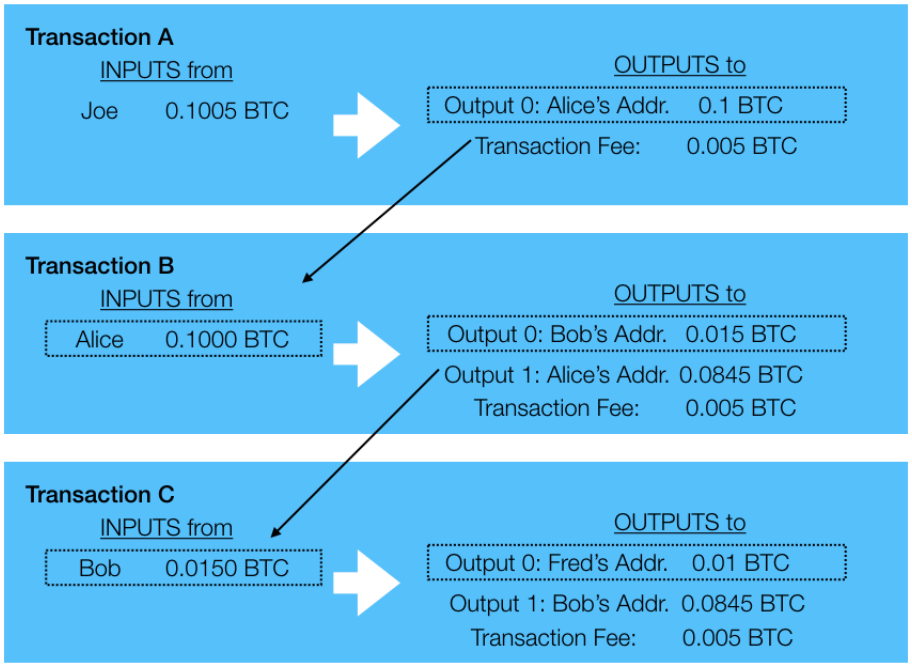

図 4.3: A chain of transactions, where the output of a previous transaction is used as an input of the next transaction

表 4.1: Structure of a Transaction

Size	Field	Description
4 bytes	Version	Specify method of processing
1-9 bytes (Variable)	Input Counter	Number of inputs
Variable	Inputs	Actual inputs
1-9 bytes (Variable)	Output Counter	Number of outputs
Variable	Outputs	Actual outputs
4 bytes	Locktime	Block number / UNIX time

Alice owns 3 Bitcoins, and if she wants to pay Bob 1 Bitcoin, she consumes one UTXO to generate outputs that transfer 1 Bitcoin to Bob and the remaining 2 Bitcoins back to herself. The transaction returns the change to the payee.

Transaction Outputs All Bitcoin transactions are listed on the Bitcoin distributed ledger. As soon as the Bitcoin network recognizes the transaction, the output is spendable in the future.

Transaction output are made of two parts:

- Bitcoin amount
- A locking script that determines the conditions to allow the spending of the output. Most scripts lock against a given Bitcoin address who becomes the new owner.

Table 4.2 contains the transaction output structure.

表 4.2: Structure of a transaction output

Size	Field	Description
8 bytes	Amount	Amount in “satoshi” (10^{-8} bitcoin)
1-9 bytes (Variable)	Byte length of locking script	
Variable	Locking-Script	Conditions for spending as described on the scri

Transaction Inputs Transaction inputs refer to existing UTXO. UTXO are specified by the transaction hash and sequence number.

Table 4.3 contains the structure of the transaction input.

表 4.3: Structure of a transaction input

Size	Field	Description
32 bytes	Transaction Hash	Reference point to UTXO of consumption
4 bytes	Output Index	UTXO sequence number starting with 0
1-9 bytes (Variable)	Byte length of unlocking script	
Variable	Unlocking-Script	Script describing the conditions of UTXO con

Transaction Script

Transaction scripts are inscribed in a reverse-polish notation language. Both locking and unlocking scripts are written in this language. When the transaction is authorized, the unlocking script is verified upon its respective locking script. Most transactions are generated as Pay-to-Public-Key-Hash.

Locking scripts known as scriptPubKey set the conditions of future consumption. scriptSig, the unlocking script, contains the corresponding digital signature.

When the transaction script is processed, the unlocking and locking scripts are merged. The former and the latter are read in order. When the script is processed without any problems, the UTXO becomes consumable.

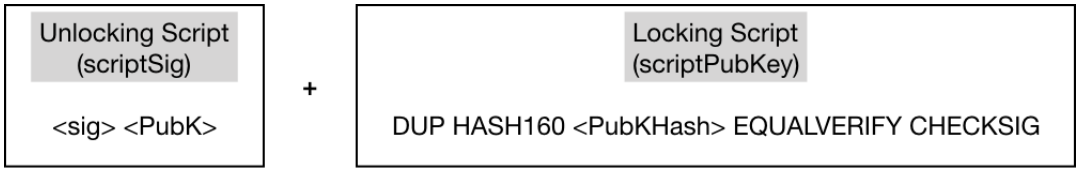

図 4.4: Combining scriptSig and scriptPubKey to evaluate a transaction script

Scripting Language

The transaction script language is a stack-based reverse-Polish notational language called Script. It has the following features.

Turing Incompleteness Script is implemented with several operators, but is limited to disable loops and advanced flow regulation. This is not Turing Complete, which makes the complexity of scripts limited and execution time predictable. Such limitations prevent "logic bombs" like infinite loops that compromise the network.

Stateless Verification Script is stateless, so there is no prior state. Therefore, information regarding script execution is self-contained. This makes script verification predictable among different nodes.

Standard Transactions

Script can be written in several ways, but Bitcoin developers basically use 5 kinds of scripts. Most nodes accept such transactions[22]:

- 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 explain about the most used kind of transaction, P2PKH.

Pay-to-Public-Key-Hash (P2PKH) Most transactions processed on the Bitcoin network is P2PKH. The locking script is notated as such:

```
OP_DUP OP_HASH160 <PubKey> OP_EQUAL OP_CHECKSIG
```

The preceding unlocking script is written as such:

```
<Signature> <PubKey>
```

Unlocking and locking scripts are combined, and if the script is verified correctly, the user can consume UTXO.

4.2.3 Blockchain Data Structure

A blockchain can be imagined as a consecutive link of "blocks". In this iamge, each block is linked to the prior block, and each block contains a data structure resembling a tree.

Each block is identified by its hash. Also, previous blocks are identified as parent blocks by their hashes. When you follow back through the links, you eventually arrive at a genesis block that started the entire blockchain.

Block Structure

Blocks contain transactions accumulated on the blockchain public ledger. The structure is labeled on Table 4.4.

表 4.4: Structure of a block

Size	Field	Description
4 bytes	Block Size	Byes
80 bytes	Block Header	Having few fields
1-9 bytes (Variable)	Transaction Counter	Transaction number
Variable	Transactions	Transaction content

Block Header

Block headers contain three kinds of metadata. The first is the hash of the previous block, the second is information regarding mining competitions including difficulty and nonce, and the third is the Merkle tree root values. The structure is listed on Table 4.5.

表 4.5: Stucture of Block Header

Size	Field	Description
4 bytes	Version	Protocol version number protocol
32 bytes	Previous Block Hash	Reference to previous block
32 bytes	Merle Root	Merkle root value from transactions within the block
4 bytes	Timestamp	Block generation time in UNIX time
4 bytes	Difficulty Target	Proof-of-work difficulty used for block generation
4 bytes	Nonce	Counter for Proof-of-work

Merkle Tree

Merkle trees, a form of data structure, efficiently summarize data of large scales and prove their integrity. Also known as binary trees, each branch of the tree contains a cryptographic hash. Hashes are calculated from below, reaching eventually to the top of the tree that contains the summary of the entire tree as the Merkle tree root value. If a Merkle tree contains N number of nodes, the node can be calculated by at most $2 * log_2(N)$ calculations.

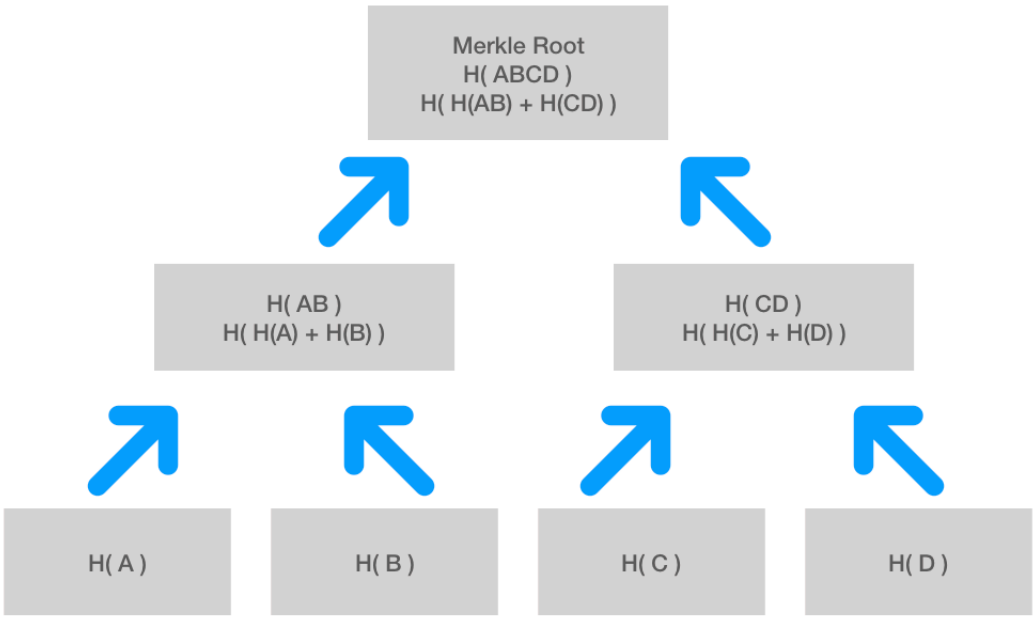

図 4.5: A merkle tree with each node hash calculated

4.3 OpenTimestamps

OpenTimestamps, developed by Peter Todd[23], is a timestamping utility that takes advantage of the Bitcoin public ledger. In comparison to similar tools, its advantages lie in:

1. Trust
   OpenTimestamps uses the Bitcoin blockchain. Bitcoin itself is a public and openly-verifiable protocol. This removes the need to trust a central authority.

2. Cost
   OpenTimestamps compresses multiple timestamps into a single transaction, so it is theoretically scalable infinitely. This high scalability is the reason why this service can be provided for free.

3. Convenience
   OpenTimestamps has no need to wait for a Bitcoin transaction to complete, generating verifiable timestamps available within a matter of seconds.

In the past, OpenTimestamps has proven its efficacy in timestamping a large volume of documents, such as storing the entirety of the Inernet Archive into a Bitcoin transaction.[24] Although alternatives exist, they don ' t aggregate document hashes and issue timestamps on a *per* document basis, leading to inefficient and expensive operations. For example, poex.io[25] bills 2.5 mBTC or 35 USD (rate as of `January 11, 2018`[26]) per document.

4.3.1 Mechanism

OpenTimestamps stores document hashes on the Bitcoin blockchain. This section explains the underlying mechanism.

Time Declaration

Bitcoin block headers contain an entry called "nTime." For a Bitcoin block to be accepted, it has to have a time entry approximately close to that of the network. The accuracy is not highly granular at all, with differences ranging from 2 to 3 hours, but can serve with enough accuracy on a day basis.

By using this property, Bitcoin can act as a notary, and Bitcoin blocks can be used for time declarations.

Merkle Trees

Merkle tree root values are contained in the header of Bitcoin blocks.

As discussed in 4.2.3, Merkle tree root values are calculated from all transactions arranged in a Merkle tree.

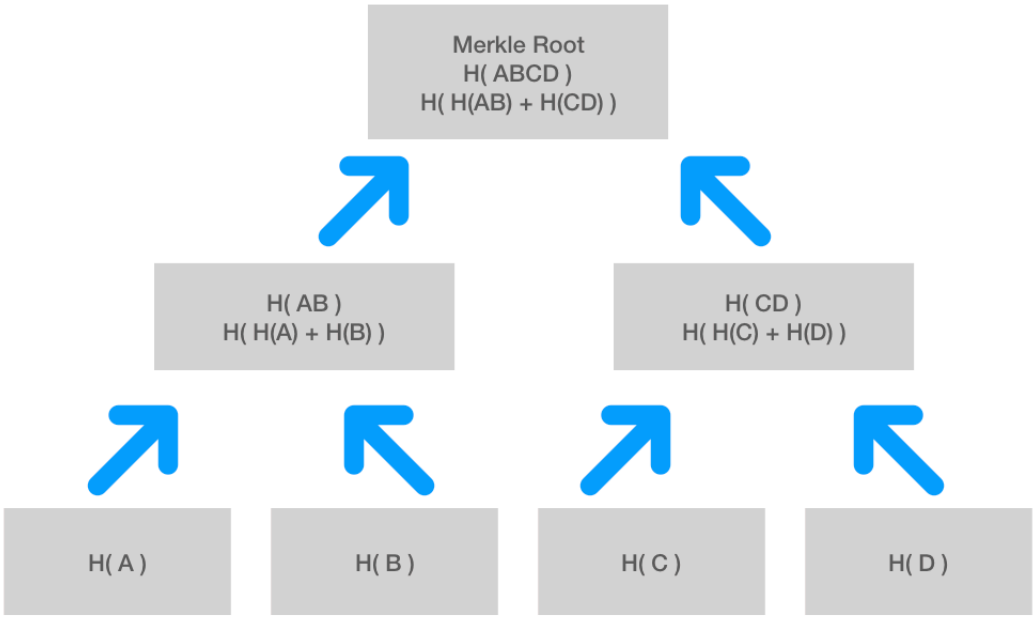

図 4.6: A merkle tree with each node hash calculated

Merkle tree root values rely on other hash values, so if even one of the underlying hash value changes, the Merkle tree root value changes as well.

To illustrate this, let's construct an example Merkle tree. For our original sequence to construct the Merkle tree, we use the following:

```
[1,2,3,4]
```

This produces the following root hash value:

```
16fbd7d1f18d2fedb247d73edc3bc6aa040f5ab99bd3b48c35b79e543d22179b
```

If we change the last entry of the above sequence to this:

```
[1,2,3,5]
```

and recalculate the Merkle tree root hash value, we get a completely different value:

```
9d671f9c9a8cc13bb2e2f2dcb6bf6a3ef8816122ab1bcd492254a05e637e0749
```

Although “hash collisions”, or complete matches of hash values from different data, can possibly occur among nodes, that possibility is astronomically small.

Commitment Operations

The commitment of block headers to Merkle trees can be considered a proof. Merkle trees commit to a transaction, and the transaction will be committed to the block header. This is because should the transaction change, so would the header.

In OpenTimestamps, the timestamp proof is a sequence of commitment operations applied to a given message. In this case, results from replaying the applied operations can prove the existence from a given time. It would be difficult to modify the message itself without changing the result and nullifying the timestamp, as each commitment operation almost certainly inserts different inputs and results. Below is an example of an operation sequence in a timestamp:

```
$ ots info 201104.csv.ots
File sha256 hash:
   bd9fd9e6bfca0f5dca6628797b9b6daa7975a5f6848e2c354d8bfc7186e5c987

Timestamp:
append b38a9f80cf3858298a99e696340156a1
sha256
 -> append 154ff59468d69840486b1864e573d63e
    sha256
    prepend 5a33e966
    append 556cdcd4f13419f8
    verify PendingAttestation('https://bob.btc.calendar.
       opentimestamps.org')
 -> append cf4a52adf1bcced55af06a286225f25a
    sha256
    prepend 5a33e967
    append 29b32cbab911f194
    verify PendingAttestation('https://finney.calendar.
       eternitywall.com')
 -> append ec57f51a2c4c5fd0786d1a0fc00573fb
    sha256
    append 76318
       b38772165f49782c78bf474a1fbe5010812e18cb4d956524e5191410d4a

    sha256
    prepend 5a33e967
    append 1a8a975c5bf9d885
    verify PendingAttestation('https://alice.btc.calendar.
       opentimestamps.org')
```

Scalability Through Aggregation

Unlike timestamping services that issue a timestamp per documents on the blockchain, Opentimestamps operates efficiently by aggregating document hashes into a single Merkle tree that is then inserted into a transaction.

To improve upon this, OpenTimestamps provides aggregation servers that allows anyone to generate timestamps from hashes. As of the time of writing, the publicly provided servers include `a.pool.opentimestamps.org`, `b.pool.opentimestamps.org`, and `finney.calendar.eternitywall.com`.

These servers can become points of centralization, but the adverse effect is kept minimal. If a server goes offline, the only inconvenience it causes is the lack of accessibility. The integrity of timestamps are proven by the Bitcoin blockchain, so servers have no way to forge 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

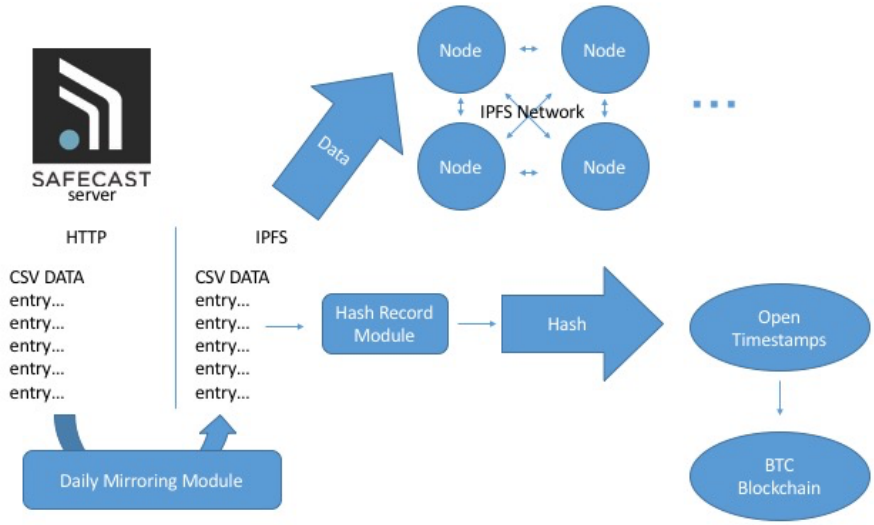

図 5.1: System Diagram

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.[27] The VPS operates using the OpenStack platform.[28] 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
```

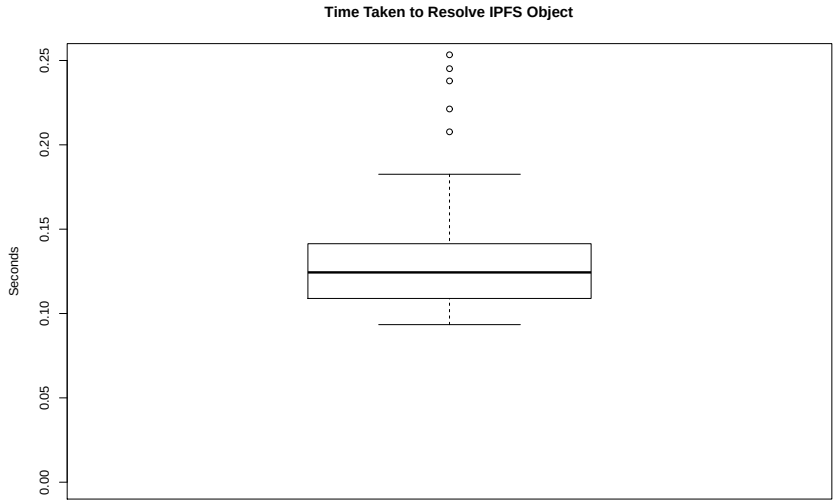

図 5.2: Evaluation Result

```
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` [29], we used version 9.0.1.

A.1.2 Pandas

Summary

Pandas[30] 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[31], an acronym for "InterPlanetary File System", is a distributed file system enabled by P2P technologies. There are client API liraries implemented in Javascript[32] and Python[33] 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[34]
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[1] 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[35] 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[36]

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 begins to connect to other nodes and download the entire Bitcoin public ledger. This ledger calls for a significant amount of storage, and exceeds 150GB as of `2018/1/4`.[37] Having the entirety of the ledger is a prerequisite for examining transactions and announcing blocks to other nodes, but is not necessary for our purpose of confirming hashes within the blockchain. Therefore, there is no problem in minimizing the storage space for the ledger via pruning.

In the Bitcoin system, there are four kinds of data:

- Raw blocks received from the network
- Undo data
- Block index database
- UTXO set database

Databases are composed from the raw data. Activating pruning will remove raw block and undo data from the local machine once integrated into the database by `bitcoind`. Afterwards, these data are used for relaying to other nodes, data reconstruction, searching old transactions, and rescanning wallets. Block indexes continue to hold metadata regarding blocks.[38]

To enable pruning, open `bitcoin.conf` and write in `prune=<N>`. N is the amount of storage provided for raw data and undo data in MiB, and the minimum requirement is 550MB.

A.2.2 Install Script

To install the script made for this thesis, you can download it from a Git repository.[39] 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[1]. 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.

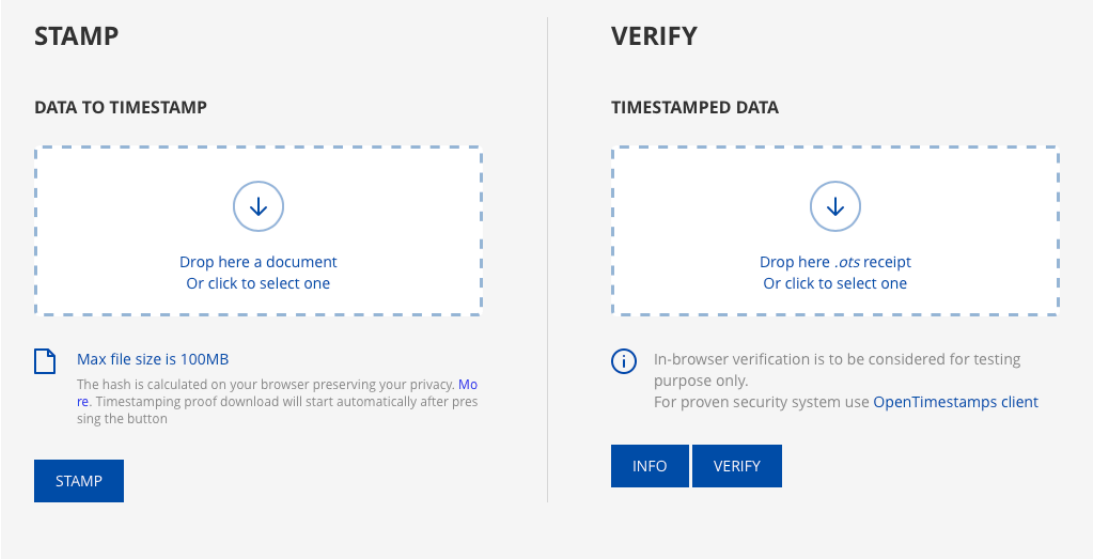

図 A.1: Online verification via opentimestamps.org[1]

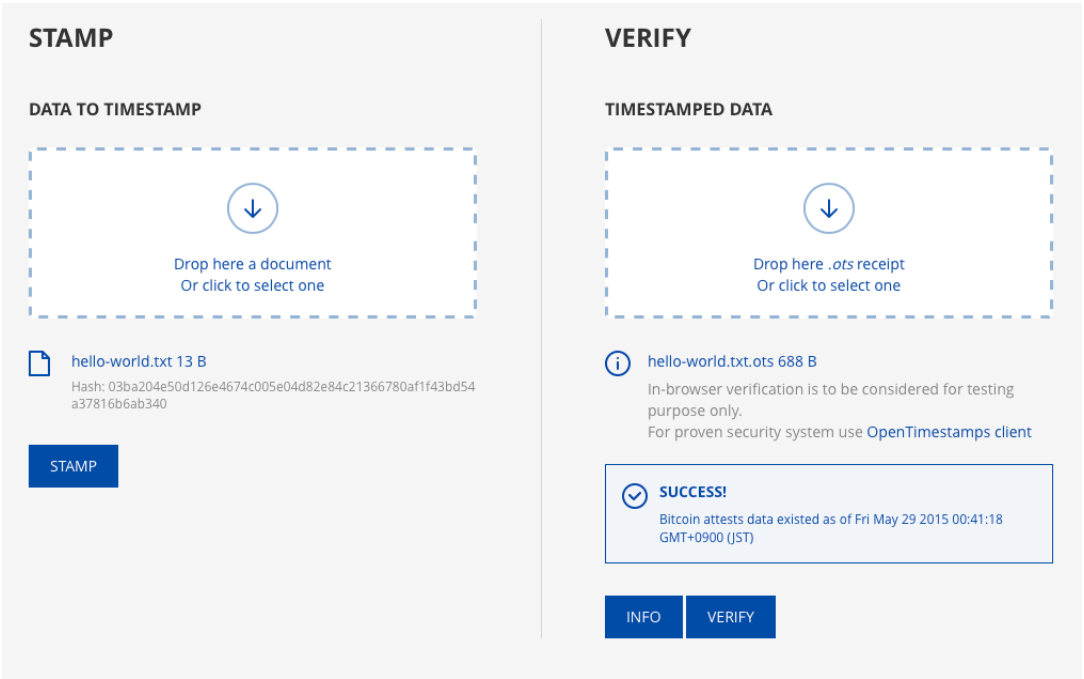

図 A.2: A successful online verification

A.3.2 Local Verification

To verify a hash locally, do the following:

1. Change to the directory containing the datasets.

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

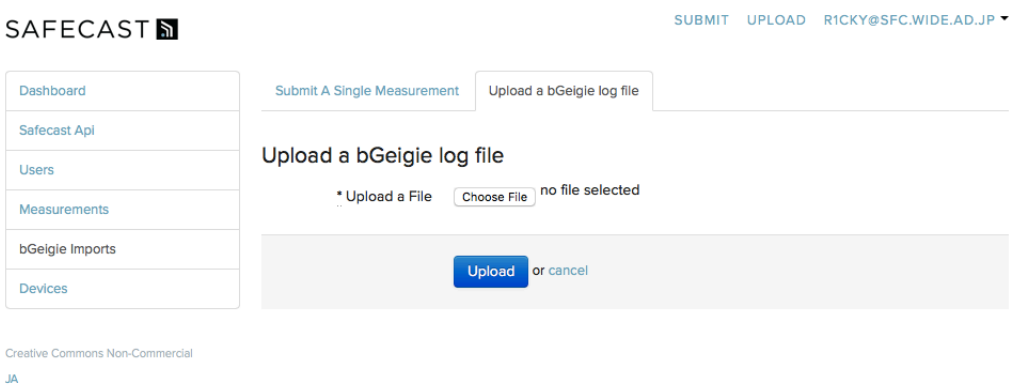

図 B.1: Uploading bgeigie data on SAFECAST

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[7] 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.

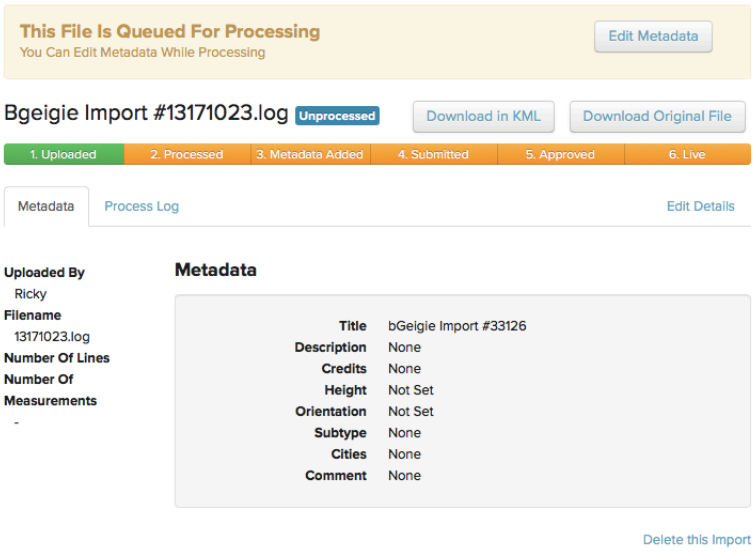

図 B.2: Screen after data has been uploaded

2. Processing

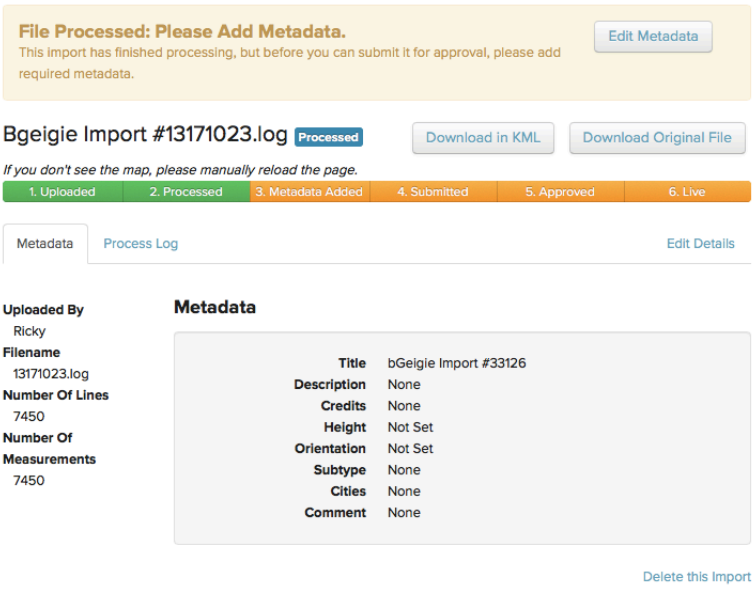

図 B.3: Screen after data has been processed

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

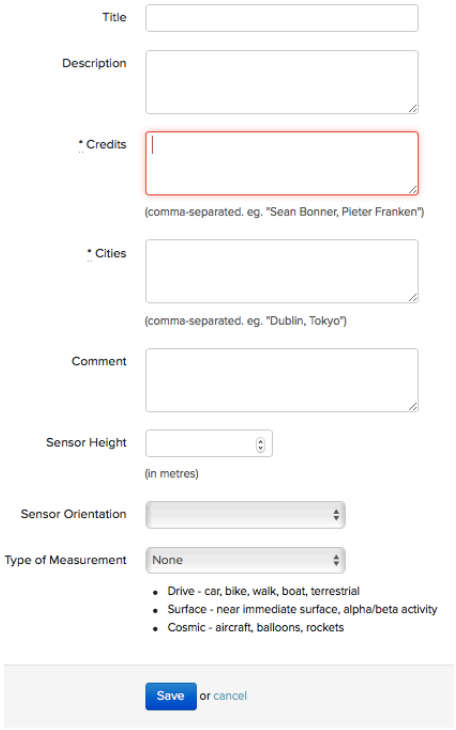

図 B.4: Prompt to enter metadata

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.

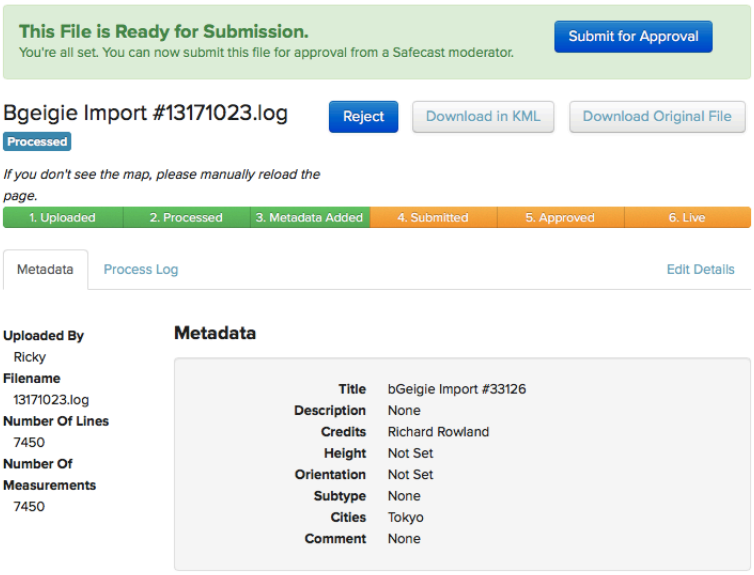

図 B.5: Screen indicates that log is ready to be submitted

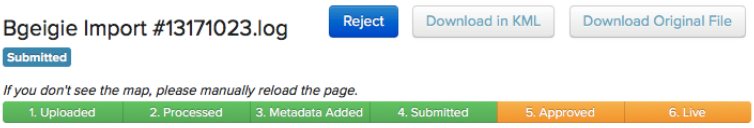

図 B.6: Screen indicating that log has been submitted for approval

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] OpenTimestamps. OpenTimestamps. https://opentimestamps.org.

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

[3] 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.

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

[5] About Safecast — Safecast.

[6] Kate Dougherty. Parts List, 2017.

[7] SAFECAST. The safecast api. https://api.safecast.org.

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

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

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

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

[12] Doozer. Doozer. https://github.com/ha/doozerd.

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

[14] HashiCorp. Consul. https://www.consul.io.

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

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

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

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

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

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

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

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

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

[24] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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