TU WIEN Informatics

Reducing the operation cost of a file fixity storage service on the ethereum blockchain by utilizing pool testing strategies.

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Bsc. Michael Etschbacher
Matrikelnummer 51828999

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.univ.Prof. Dr. Andreas Rauber

Wien, 19. Mai 2022

Michael Etschbacher	Andreas Rauber

TU WIEN Informatics

Reducing the operation cost of a file fixity storage service on the ethereum blockchain by utilizing pool testing strategies.

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Bsc. Michael Etschbacher
Registration Number 51828999

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.univ.Prof. Dr. Andreas Rauber

Vienna, 19th May, 2022

Michael Etschbacher	Andreas Rauber

Erklärung zur Verfassung der Arbeit

Bsc. Michael Etschbacher

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. Mai 2022

Michael Etschbacher

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Enter your text here.

Contents

CHAPTER 1

Introduction

1.1 About Data Integrity

Storing cultural heritage in digital archives offers malicious actors the possibility to manipulate the data and possibly forge history. Recent digital technologies make data manipulation more efficient, less costly, and more exact and there is a long history of forging history. In 1920 a photography was taken of Vladimir Lenin atop a platform speaking to a crowd. In the original photo, see Figure 1.1a, Lenin's comrade Leon Trotsky can be seen standing beside the platform on Lenin's left side. When power struggles within the revolution forced Trotsky out of the party 7 years later, he was retouched out of the picture, see Figure 1.1, using paint, razors and airbrushes. Soviet photo artists altered the historical record by literally removing Trotsky from the pictures [HS05, 3].

(a) Original

(b) Original

Figure 1.1: Catalog Images

Digital archives must earn the trust of current and future digital creators by developing a robust infrastructure and long-term preservation plans to demonstrate that the archive and its staff are trustworthy stewards of the digital materials in their care [KORD10,

37]. Digital objects can be corrupted easily, with or without fraudulent intent, and even without intent at all. Data corruption is usually detected by comparing cryptographic hashes, so-called fixity information, at different time intervals [DFG+14, 1]. The object is seen as uncorrupted if the hash values are identical, since the smallest change to the object would alter the newly computed hash value immensely.

1.2 Research Problem

Although generating fixity information (e.g., MD5, SHA-256) is relatively easy, managing that information over time is harder considering that if a malicious actor can alter the fixity information, the actor is also able to alter the underlying object illicit [KORD10, 35]. Fixity information is usually stored in databases; object metadata records or alongside content, whereas this thesis deals with blockchain as a storage medium. [CBB+18] and [SBP+20] have shown that the Ethereum blockchain, implemented by [B+13], can indeed be utilized to ensure data integrity, but there is a problem with that: the operation cost. The cost of storing a SHA256 values on the Ethereum blockchain is 20,000 gas (the unit for transaction fees), which oscillate at the time of writing at about $5, which means the operation cost for an ingest of 10,000 objects costs about $50,000.

1.3 Research Goal

This thesis proposes a way to reduce the operational cost of a blockchain-based fixity information storage by applying a pool testing strategy in which several digital objects are combined in a pool to form a hash list. The idea for this approach stems from the ongoing pandemic, in which the test capacities also must be used optimally.

> Pool testing strategies build on testing a pooled sample from several patients: if the results from the pool test are negative, all patients in the pooled sample are declared not to have COVID-19; if the results of the pool are positive, each patient sample is tested individually. The pooled testing strategy is appealing, particularly when test availability is limited [CGWK20, 1].

This paradigm will be utilized in this thesis where the test specimen are cryptographic hashes, and the pool is hash list.

1.4 Research Questions

Based on the problem described above, the following research questions arise:

RQ1 What is the optimal pool size based on the corruption rates of digital objects in the archive in terms of efficiency and cost

RQ2 To what extent can pooled object hashes increase the efficiency and reduce cost for a fixity information storage service on the Ethereum blockchain?

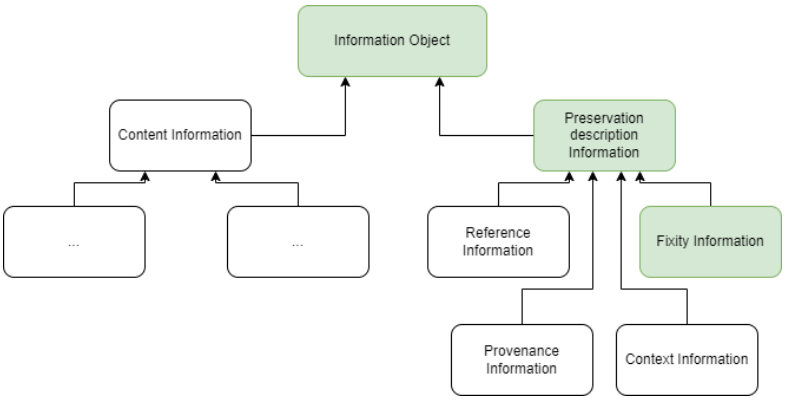
RQ3 Given that metadata has a higher corruption rate, what effect has the split of metadata and objects on the operation cost?

1.5 Research Approach

This work will follow the principles of Design Science Research, from [Hev07]. The artifact is designed based on a literature review of the following main topics *Diplomatics of Digital Records*;*Ethereum Blockchain*; and *Pooled testing*.

The *Application environment* is a digital archive which manages any kind of data, data will be referenced as information object in the archive and each object is provided with necessary metadata, as described in the OAIS reference model. I assume that the data integrity is ensured on ingest where a cryptographic hash value of the information object is computed and stored on the blockchain. The "location" of the fixity information on the blockchain is stored in the preservation description information in the form of a unique transaction hash, see Figure 1.2.

Figure 1.2: The transaction hash, used to retrieve data from the blockchain, is stored in the fixity information of the PDI of an information object [Lee10, 7]



The *Artifact* is a combination of decentralized application on the Ethereum blockchain and a local pool structure in the archive. The smart contract exposes functions for managing cryptographic hashes of pools. The artifact must provide the following functionalities:

- Create, Read, Update SHA256 values of pools.
- Must hold a reference (ID) for the respective pool in the archive.
- Must be tamperproof for unauthorized calls.

The smart contract complements the local pool structure in the archive where fixity information of certain object pools may be requested on any given time. The artifact should reduce the amount of operations needed to ensure the integrity of 10,000 objects from ingest to retrieval process by at least 50%.

CHAPTER 2

Related Work

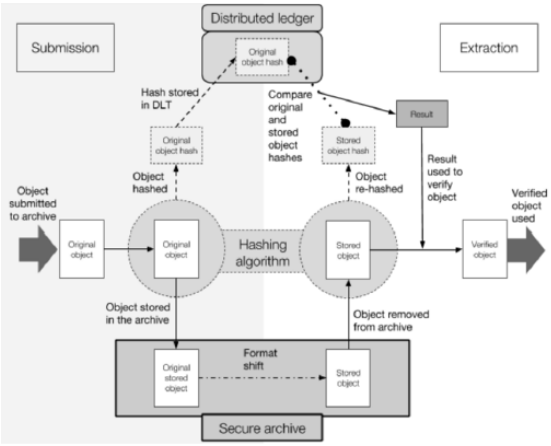
2.1 Project Archangel

Project ARCHANGEL was a decentralized fixity information developed in the UK with the participation of three other countries, Australia, Estonia and Norway. Their approach is to store a cryptographic hash of every incoming object of the archive on a proof-of-authority blockchain. The project started in 2017 and ended in August 2018, the goal of the project was to ensure the integrity and authenticity of digital objects in archives with the usage a private fork of the Ethereum network. Their design philosophy of operating a private blockchain has the flaw of giving a few entities the power to alter data on the blockchain. Currently, their implementation uses smart contracts as a gateway for writing to the Blockchain [CBB+18, 4].

The authors of the project ARCHANGEL stated that the trust conferred by the public to the Archives and Memory Institutions (AMIs) had eroded due to the ease by which forgery and unauthorized modifications to electronic records were conducted owing to advances in technology and the generation of numerous types of composited content. To recover public trust in AMIs and guarantee the integrity of the records, a method utilizing the existing databases as well as a method to utilize a merkle tree or durable storage provided by private corporations have been taken into consideration; however, it was concluded that they are difficult to apply owning to several limitations. Unlike in the past, when archives have relied on the products and technologies of certain companies, blockchain presents a completely different paradigm from openness and expandability. The blockchain awards permission to write records in a distributed ledger to only authorized institutions, whereas permission to view the recorded content after accessing the distributed ledger is granted to every node participating in the blockchain. Moreover, scalability allows the utilization of various tools provided by open-source codes and enables the integrity of records to be assured by multiple parties through a consensus mechanism rather than by a single centralized institution [WY21, 4].

The proposed architecture of ARCHANGEL can be seen in Figure 2.1 where the cryptographic hash of an incoming object is computed and then persisted onto the private blockchain. After a certain time interval, the object can be retrieved from the archived and a hash will be recomputed with the same cryptographic hash function, the object is guaranteed to be unaltered if the local and online hash value are the same. I agree with

Figure 2.1: Architecture of the ARCHANGEL platform [CBB+18, 2]



their vision of publicly available fixity information, where everyone with an Ethereum client is able to validate the integrity of objects in archive, but there are two points which do not conform with my vision. First and foremost is the usage of a private fork of the Ethereum network that is operated by a private set of nodes, which is basically a proof-of-authority consensus mechanism and contradicts the vision of a decentralized blockchain [CBB+18, 3]. Second, they have not implemented some form of transaction reduction, therefore they have to upload a SHA256 word for each object in the archive, which is not feasible since a single upload costs about $8, see Section 4.8.

2.2 Provenance framework for the IOT

Sigwart et al. 2020 proposed a generic data provenance framework for the IoT. Their approach is to store provenance records on the Ethereum blockchain with the goal of immutable provenance records for supply chains or sensor data [SBP+20, 4]. It is implemented as a framework, therefore it can be adapted to suit other forms of data, in my case fixity information. Their implementation utilized the ERC727 token standard for Ethereum which is beneficial for provenance data, where each object is a non-fungible-token(NFT) [SBP+20, 7]. NFTs require a large amount of metadata, which is costly and for my use-case certainly overhead.

2.3 Remote Data Integrity Checking Scheme

Zhao et al. 2020 proposed a blockchain based remote data integrity checking scheme. Their scheme consists of three phases, including setup, storage and verification phase, this scheme may be of great importance for my work because it integrates well with the work of Sigwart et al. 2020 in terms of design goals. The detail of each phase in their scheme is described as follows: (1) Setup phase, where the public parameter of the elliptic curve group is uploaded to the blockchain. (2) Storage phase, where the data owner signs the data blocks and uploads them to a cloud storage. (3) Verification phase, where the data owner downloads the data or desires to check the integrity of the individual data. The auditing process is done by verifying the signature of the data blocks by the data owner [ZCL+20, 1].

2.4 Pooled testing

Pooled Testing was first introduced by [Dor43] as a strategy to screen numerous military recruits for syphilis during World War 2. Dorfman envisioned that instead of testing each recruit's blood specimen separately, multiple specimens could be pooled together and tested at once. Positive pools' specimens would be retested individually to determine which recruits had contracted the disease, whereas negative pools' specimens would be declared negative. Dorfman wanted to save money on testing while still identifying all syphilitic-positive candidates, so he used group testing. Because the goal is to identify all positive persons among all individuals tested, this is now known as the "case identification problem." Dorfman's case identification method can be thought of as a two-stage hierarchical algorithm. Individuals from positive pools are tested in the second stage after non-overlapping pools have been screened in the first. When the level of corruption is low, higher-stage algorithms have proven to be effective in lowering the number of tests required. [HTBM17, 1]. In this thesis, two strategies of pooled testing are implemented. First, two stage hierarchical pooling strategy, where in the first stage of this protocol N pools get initialized and filled with samples of the population. If the combined result of the pool is negative than no second stage is needed, but when a pool is declared positive a second stage is needed and all individuals from this pool have to be retested in order to find the corrupted individual. The expected number of test is equal to the number of tests in the first stage added to the number of tests in the second stage [NEG+21, 3]. Second, context-sensitive pooling, where homogeneous pool samples are grouped where it is assumed that the within-group variation in any characteristic is smaller than the between-group variation. Hence, if one member of a pooled group is corrupted, there is a high likelihood that other group members are also infected [DBK20, 3].

2.5 Wrap up

The blockchain related projects presented in this Chapter have a common vision, which is a decentralized storage for metadata to validate the integrity and authenticity of data used in applications. The ability for the public to validate that an authority did not tamper with the data in their care is important and the removal of trust strengthens the trust in public institutions. More on the subject of trust can be seen in Chapter 3 Decentralization and trust aside, the cost of the fixity storage presented in this thesis is also more than relevant as later seen in Chapter 4, where the cost and computational effort of interacting with the Ethereum network is presented.

CHAPTER 3

Diplomatics of Digital Records

3.1 About Diplomatics

Diplomatics is a discipline founded in France in the seventeenth century by Benedictine monk Dom Jean Mabillon in his dissertation De Re Diplomatica Libri VI (1681) to determine the provenance and authenticity of evidence attesting to patrimonial rights. It was then utilized by attorneys to settle disputes, historians to interpret documents, and editors to publish medieval deeds and charters, and it evolved into a legal, historical, and philological specialty. Classic diplomatics and modern/digital diplomatics should be distinguished since these two areas of the field do not represent a natural evolution of the latter from the former, but rather operate in parallel and focus on separate topics. Modern diplomatics encompasses all documents produced in the course of any kind of business, and is defined as "the discipline that studies the genesis, forms, and transmission" of records, as well as "their relationship with the facts represented in them and with their creator, in order to identify, evaluate, and communicate their true nature." Both classic and modern diplomatics are concerned with determining the trustworthiness of records; however, the former does so retrospectively, examining records from several centuries ago, whereas the latter is concerned not only with establishing the trustworthiness of historical information but also with ensuring the trustworthiness of records yet to be created [KORD10, 10].

A digital record is a digital component, or group of digital components, that is saved, treated, and managed as a record, or, more specifically, a record whose content and form are encoded using discrete numeric values, such as the binary values 0 and 1, rather than a continuous spectrum of values, as defined by modern diplomatics. A digital record differs from analogue and electronic records in that it is digital. The representation of an item or physical process using continuously varying electronic signals or mechanical patterns is referred to as analogue by InterPARES. An analogue depiction of an object or physical process, in contrast to a digitally recorded representation, closely mimics

the original. Any analogue or digital record conveyed by an electrical conductor and requiring the use of electronic equipment to be comprehensible by a person is defined as an electronic record by InterPARES. Using the classic archive idea, InterPARES defines a record as a document created or received as an instrument or by-product of a practical activity and set aside for action or reference [Dur09, 52].

The integrity of a record is determined not only by its appearance – which can be deceiving in the case of good forgeries – but also by the circumstances of its maintenance and preservation: until proof to the contrary, an unbroken chain of responsible and legitimate custody is considered an insurance of integrity, and integrity metadata are required to attest to that. The duty for a record's authenticity changes from the creator's trusted record keeper, who must guarantee it for the duration of the record's custody, to the trusted custodian, who must guarantee it for the duration of the record's existence [Dur09, 53]. In this thesis the Ethereum network acts as a trusted custodian which guarantees that no one is able to tamper with the fixity information of digital records.

3.2 Trustworthiness

Whether it's a stack of paper or a WordPerfect file, trustworthiness is a concept and an obligation that lasts the life of the document. As files go through the stages of the preservation process, from initial capture and metadata extraction to longer-term strategies like migration and rights management, the needs of born-digital objects change. Born-digital fonds follow a similar path from creator to intermediary, such as a dealer or other (human or technology) agent, to archival repository staff, and eventually to storage and, possibly, ingest into a digital repository. The phases of that journey make up a digital object's chain of custody, and each one has a significant impact on the trustworthiness of the born-digital elements in a given accession. The veracity of a document's content is often not a concern of archivists working with cultural heritage materials, and although reliability is an important component of trustworthiness, the veracity of a document's content is often not a concern of archivists working with cultural heritage materials. The fundamental parts of creating and maintaining trust are the provenance of both analog and digital resources, as well as documentation concerning their storage environment, what has been done to them, and by whom. In the collection and preservation of born-digital resources, the trustworthiness of an institution, a custodian, or a document is critical [KORD10, 27]. Modern diplomatics concerns itself with five aspects of trustworthiness: reliability, authenticity, accuracy, integrity and authentication [KORD10, 10].

3.2.1 Reliability

Reliability is the trustworthiness of a record as a statement of fact, as to content. It is assessed on the basis of 1) the completeness of the record, that is, the presence of all the formal elements required by the juridical-administrative system for that specific record to be capable of achieving the purposes for which it was generated; and 2) the

controls exercised on the process of creation of the record, among which are included those exercised on the author of the record, who must be the person competent, that is, having the authority and the capacity, to issue it. The reliability of a record is the exclusive responsibility of its creator and the trusted record keeper, that is, of the person or organization that made or received it and maintained it with its other records [Dur09, 52]. It is evaluated on the basis of 1) the completeness of the record, that is, the presence of all formal elements required by the legal-administrative system for that specific record to be capable of achieving the purposes for which it was created; and 2) the controls exercised on the process of creating the record, among which are those exercised on the author of the record, who must be the person competent, that is, having the authority and capacity to create the record. The reliability of a record is the exclusive responsibility of its creator and the trusted record keeper, that is, of the person or organization that made or received it and maintained it with its other records, therefore it is important that public archives gain the trust of the public to be considered as a reliable steward of public cultural heritage [Dur09, 52]. Open source software is asked because community members can look into the code and verify that no unwanted actions will be executed. The blockchain, in regard to reliability, is well accepted by the public and every transaction ever made is publicly viewable. Also, the source code of the Ethereum network and each update (fork) gets peer reviewed and is available on GitHub[1].

3.2.2 Authenticity

Authenticity refers to a record's ability to be trusted as a record, and is defined as the fact that a record has not been tampered with or corrupted, either accidentally or with malicious intend. An authentic record keeps the same identity it had when it was created and can be presumed or demonstrated to have kept its integrity across time. The identity of a record is made up of all the features that set it apart from other records, and it is determined by the formal components on the record's face and/or its attributes, as stated in a register entry or as metadata [Dur09, 52]. Archivist' expectations of source reliability have evolved over time, from the belief that librarians and archivists would provide researchers with verifiable evidence, to more modern understandings that reject the ideal of the reliable source and regard all texts as potentially deceptive and richly ambiguous. Data-stewards should be able to offer researchers with documentation about the provenance and acquisition of the data in their care using the methods of operation and processes created by the community and professionals, such as a provenance chain or the fixity information created on ingest of the object in question [KORD10, 32].

3.2.3 Accuracy

Because record correctness was subsumed under both trustworthiness and authenticity as a notion, it was never a consideration in general diplomatics. Accuracy is defined as the truthfulness, exactness, accuracy, or completeness of the data (i.e., the smallest,

[1] `https://github.com/ethereum/go-ethereum`

meaningful, indivisible pieces of information) within a record. Because of the ease with which data can be damaged during transmission over space (between humans and/or systems) and time in the digital environment, accuracy must be considered and assessed as a separate quality of a record (when digital systems are upgraded or records are migrated to a new system). As a result, the responsibility for accuracy shifts over time from the creator's trusted record-keeper to the trusted custodian [KORD10, 14].

3.2.4 Integrity

There are no originals in the diplomatic sense in the digital environment, that is, there are no records that are the first instance of each item under consideration, in addition to being complete and capable of reaching the purposes for which they were generated, because when we close a digital record for the first time, we destroy the original and every time we open it, we create a copy. However, we can state that each digital record is a copy in the form of original in the latest version utilized by the creator in the usual and ordinary course of business, and that any version retained by the preserver is a genuine copy of the creator's record. If their identity is intact and their integrity can be presumed or demonstrated, they are both authoritative and authentic. When extracting digital evidence, digital forensics must first avoid modifying the data and must be repeatable in order to be considered reliable. The accurate documenting of each and every operation performed on the evidence supports repeatability, which is one of the key axioms of digital forensics practice [Dur09, 58]. Such a chain of alteration is possible on the Ethereum network, since each update on an object hash is natively documented on the blockchain. Duplication integrity is ensured when given a data set, the process of creating a duplicate of the data does not modify the data (either intentionally or accidentally) and the duplicate is an exact bit copy of the original data set. It is possible to preserve data integrity over the duplicate, with respect to the original, by using a trusted third party. At the time the image is created, a copy of the hash can be given to a trusted third party to hold in escrow. Now changes to the duplicate can be detected even if the original is modified. Experts in digital forensics have also linked duplication integrity to time, and have investigated using time stamps to achieve this. A distinction between the integrity of a record as a whole and the integrity of its duplicate may be useful in resolving the conflict between diplomats' and information technology experts' views on integrity, which tend to support the need for the extreme authentication provided by a digital signature. Indeed, by accepting the digital forensics specialists' proposed link between integrity and time, and defining record integrity differently in each step of the record life cycle and/or custodial history, one could further enrich the concept of integrity [Dur09, 60]. Both diplomatics and forensics were created as methods for examining existing material evidence, determining its transmission status, legitimacy, and ability to offer confirmation of the circumstances at hand [Dur09, 64].

3.2.5 Authentication

Authentication is defined as a statement or an element, such as a seal, a stamp, or a symbol, attached to a record after it has been completed by a competent officer. Authentication simply assures that a record is authentic at one precise point in time, when the declaration is made or the authenticating element or entity is affixed, whereas authenticity is a quality of the record that accompanies it for as long as it remains as is. A digital signature is frequently used in the digital environment to give ultimate authenticity. Because it is tied to a full record, the digital signature serves as a seal, allowing verification of the record's origin and integrity, as well as making the record unassailable and incontestable by performing a non-repudiation function. [Dur09, 53]. With regard to authentication, the method presented in this thesis guarantees that the fixity information of an object is never altered unnoticed by an unauthorized actor, since every transaction can be viewed attached with a timestamp on the blockchain.

3.3 Wrap up

The concept of trust in a public authority is presented in this Chapter and acts as a foundation for designing the fixity storage. Concepts, such as integrity and accuracy are implemented in the form of SHA256 hashes, with which the integrity of an object can be validated at bit level. Reliability is enforced by the nature of the blockchain, where every state changing action is documented immutable as long as the blockchain exists, as later presented in Chapter 4. Each of the core concepts of trust is needed in the modern digital world, where it is easier than ever to manipulate data and opinions.

CHAPTER 4

Ethereum Blockchain

4.1 Introduction to Ethereum

Open source blockchain networks like Ethereum and Bitcoin are software kits that allow you to create an economic system in software, replete with account management and a native unit of exchange that can be used to transfer funds between accounts. These native units of exchange are referred to as coins, tokens, or cryptocurrencies, but they are the same as tokens in any other system: they are a type of money that can only be used within that system to pay peers or run programs on the Ethereum network. When you want to make one of these peer-to-peer networks accessible through a web browser, you need to use special software libraries such as Web3[1] which is utilized in this thesis to connect to the Ethereum network [Dan17, 2]. The fact that everyone in the world can interact with the same distributed database, given you have internet access and a decent device is favorable for a fixity storage such as presented in Chapter **??**, since anyone will be able to confirm if an object has changed over time if given the right resources, such as the transaction hash in which the fixity information of an object was stored on the blockchain. Let us picture a digital archive which releases multiple objects to the public and their respective fixity information on the blockchain, the community could then validate if the object in question is authentic. Pure trust, that an archive or its stewards did not tamper with the data in their care is therefore not needed anymore.

4.2 About the blockchain

A block is a time unit that contains a specific amount of transactions. Transaction data is captured throughout that time period, and when the unit of time expires, the next block begins. The blockchain is a representation of the history of state changes in

[1] `https://github.com/ethereum/web3.py`

the EVM's network database [Dan17, 43]. The Ethereum network divides transactions and state changes into blocks, which are then hashed. Before the following canonical block may be added on top of it, each block is inspected and validated. Nodes on the network will no longer need to evaluate the trustworthiness of every single block in the Ethereum network's history; instead, they will only need to compute the current balances of the accounts on the network. They only check to see if its "parent block" is the most recent canonical block. They do it rapidly by checking to verify if the new block has the right hash of its parents' transactions and state. The blockchain is made up of all the blocks strung together, including the genesis block, which is an honorific designating the first block mined by the network after it went live. The blockchain is also known as a distributed ledger or distributed ledger technology in some sectors (DLT). The term "ledger" is correct since the chain contains every transaction in the network's history, thereby turning it into a massive, well-balanced ledger [Dan17, 55].

4.3 Ethereum Virtual Machine

The physical manifestation of the EVM cannot be compared to that of a cloud or an ocean wave, but it does exist as a single entity sustained by thousands of connected computers running an Ethereum client[2]. In the Ethereum context, a virtual machine (VM) is a massive global computer made up of constituent nodes, which are themselves computers. In general, a virtual machine is a computer system that is emulated by another computer system. These emulations are based on the same computer architectures as the target of their emulation, but they usually reproduce that architecture on hardware other than that for which it was designed. Virtual machines can be built using hardware, software, or a combination of the two. It is both in the case of Ethereum. Rather than securing thousands of individual machines, Ethereum takes the approach of securing one massive state machine that can span the entire globe [Dan17, 48]. The EVM can run any computer program written in Solidity[3], the native programming language of the EVM. These programs will always create the same output, with the same underlying state changes, when given a specific input. As a result, Solidity programs are entirely deterministic and guaranteed to execute, as long as you pay enough for the transaction. Solidity programs are Turing complete in the sense that they can express all tasks that can be performed by computers. This means that every application executed on the platform is run by the entire dispersed network, every node [Dan17, 50]. The EVM is also a runtime environment for small programs that can be executed via a network, from the perspective of a software developer. Smart contracts are compiled into the EVM's native language, the EVM bytecode. By using a client application like Truffle[4] or Geth[5], Solidity, a high-level language, is compiled into bytecode and posted onto the Ethereum

[2] https://ethereum.org/en/developers/docs/evm/
[3] https://github.com/ethereum/solidity
[4] https://trufflesuite.com/
[5] https://geth.ethereum.org/

network [Dan17, 51]. I have decided to utilize Truffle for this thesis because of its fast and easy way to interact with the Ethereum network and its subnetworks.

4.4 Smart Contracts

Smart contracts, similar to the concept of classes in traditional object-oriented programming, are the building blocks of decentralized apps operating on the EVM. When developers talk about developing smart contracts, they're usually talking to writing code in the Solidity programming language for execution on the Ethereum network. Units of value can be exchanged as readily as data when the code is run [Dan17, 10]. In my thesis, the fixity storage is often referred to as decentralized application (DAPP) which is built with several smart contracts. The smart contract implement in this thesis is further described in Section 5.3.

4.5 Persistence

Decentralized storage systems, unlike centralized servers managed by a single corporation or organization, are made up of a peer-to-peer network of users that each hold a share of the overall data, resulting in a resilient file storage sharing system. These can be found in any peer-to-peer network or in a blockchain-based application. Ethereum is a decentralized storage system in and of itself, and it is when it comes to code storage in all smart contracts. When it comes to massive amounts of data, however, Ethereum was not intended for that. The chain is steadily growing and on March 19, 2022, the Ethereum chain is at 602.95 GB[6], which every node on the network needs to be able to store. If the chain were to expand to large amounts of data (say 5TBs) it wouldn't be feasible for all nodes to continue to run. Also, the cost of deploying this much data to Mainnet would be prohibitively expensive due to gas fees[7]. My proposed solution utilizing pooled testing, presented in Chapter **??**, mitigating the ever-increasing chain size of the Ethereum network by reducing the amount of transactions needed to operate the fixity storage. All transactions in Ethereum are stored on the blockchain, a canonical history of state changes stored on every single Ethereum node [Dan17, 12]. A blockchain has a schema, much like any other database: rules establish, constrain, and enforce relationships between entities. Motives to break or alter these relationships can be found in a wide range of industries, leading to bribery and corruption and making blockchain's trustless properties even more appealing to businesses than previous generations of software and networking. Shared read/write access generates considerable complexity in all databases. Depending on where the database is physically situated, machines all over the world may experience variable latency, resulting in certain write operations arriving out of order. This becomes even more challenging when numerous parties are expected to share a database equally [Dan17, 20]. The nodes process the block in question and execute any

[6] https://ycharts.com/indicators/ethereum_chain_full_sync_data_size
[7] https://ethereum.org/en/developers/docs/storage/

code included within the transactions. This is done independently by each node; it is not only highly parallelized, but also very redundant. Despite the significant redundancy, this is a reliable and efficient technique to balance a worldwide ledger [Dan17, 50].

4.5.1 Blockchain-based persistence

This type of persistence is utilized in my thesis, since the fixity information of the archived objects are meant to be persisted for long term. Ethereum requires a persistence mechanism in order for a piece of data to last indefinitely. When executing a node, for example, the persistence method requires that the entire chain be considered. The chain continues to grow as new pieces of data are added to the end, requiring each node to reproduce all the embedded data. The term for this is "blockchain-based persistence." The problem with blockchain-based persistence is that the chain could grow to be far too large to maintain and retain all the data in a reasonable amount of time (e.g. many sources estimate the Internet to require over 40 Zetabytes of storage capacity). In addition, the blockchain must have some sort of incentive system. For blockchain-based persistence, the incentive system includes a payment made to the miner, where they are paid to include data into the next block of the chain[8].

4.5.2 Contract-based persistence

Data used in smart contracts cannot be duplicated by every node and preserved indefinitely, hence contract-based persistence assumes that data must be kept using contract agreements. These are agreements reached between numerous nodes to hold a piece of data for a set amount of time. They must be refunded or renewed after they expire in order for the data to be preserved. Instead of storing all data on-chain, the hash of where the data is situated on the chain is often stored. This eliminates the requirement for the entire chain to scale in order to keep all the data[9].

4.6 Test Networks

These networks are not really free of charge, but the ETH token on the respective network is free to get. Although the queue of the free ETH token can be very vast in some cases, e.g. `https://faucet.dimensions.network/` this faucet has a waiting queue for over three days until you receive one ETH token for the Ropsten network. Other faucets, a distributor of free ETH test tokens, require social network accounts to verify that you are not a bot e.g. `https://faucet.paradigm.xyz/` requires a Twitter account with at least one tweet and at least 15 retweets before you can request one ETH token. The faucet request are almost all time gated, meaning that you could request ETH only once per day. At small scale, this limitation is no problem for this thesis because persisting around 10 pools on the blockchains require at max 0.002 ETH tokens. The

[8] `https://ethereum.org/en/developers/docs/storage/`
[9] `https://ethereum.org/en/developers/docs/storage/`

one per day paradigm gets problematic for the experiment for the 10,000 objects, which costs about 20 ETH tokens, which means I would have to request tokens for 20 days straight to perform one experiment on a real live environment such as the Ropsten testnet. There are multiple options to choose from for a testnet, from the most prominent networks presented at the official Ethereum documentation [10] only the Ropsten network implements the proof-of-work algorithm, which means it is the best representation of Ethereum to date, and for that reason I decided to utilize the Ropsten network in my thesis.

4.7 Gas and Fees

Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network, this property allows me to calculate the operation cost for the fixity storage without taking the price fluctuation of ETH into account. Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas fees are paid in Ethereum's native currency, ETH (ETH). Gas prices are denoted in gwei, which itself is a denomination of ETH - each gwei is equal to 0.000000001 ETH. For example, instead of saying that your gas costs 0.000000001 ether, you can say your gas costs 1 gwei. The word 'gwei' itself means 'giga-wei', and it is equal to 1,000,000,000 wei. Wei itself is the smallest unit of ETH[11]. Miners are paid in ETH for mining, and also for running scripts on the network. The cost associated with electricity expenditure of servers running on the Ethereum network is one of the factors that gives ETH, as a cryptocommodity, its intrinsic value—that is, someone paid real money to their electricity company to run their mining machine. Specialized mining rigs, which use arrays of graphics cards to increase their odds of completing a block and getting paid [Dan17, 12]. Mining creates the consensus needed to make legitimate state changes, and miners are compensated for their contributions. This is how Ethereum (ETH) and Bitcoin (BTC) are created [Dan17, 57]. To ensure that the system is not clogged by meaningless spam contracts, there must be a fee associated with each instruction the EVM executes. An internal counter maintains account of the fees incurred each time an instruction is executed, which are then charged to the user. When a user initiates a transaction, a tiny percentage of the user's wallet is set aside to pay these fees [Dan17, 58]. The costs are the driving factor of my thesis where I try to make as fewer transactions as possible to run a fixity storage on the Ethereum network. The fees are determined by the transaction's gas cost; gas is a unit of effort, not a subcurrency, therefore it cannot be held or hoarded. In computer words, it simply assesses how much work each step of a transaction will take. You only need to add ETH to your account to be able to pay for gas. There is no need to buy it individually because there is no gas token. Every operation on the EVM has a gas cost attached with it. Gas prices ensure that network computing time is adequately priced [Dan17, 59]. The only person who suffers if you transmit a set of computationally difficult commands to the EVM is

[10] https://ethereum.org/en/developers/docs/networks/

[11] https://ethereum.org/en/developers/docs/gas/

you. When the amount of ETH you allocated to the transaction runs out, the work will halt. It has no bearing on the transactions of others. There is no method to jam the EVM without spending a significant amount of money in transaction fees. The gas price system serves as a de facto means of scalability. Miners have complete control over which transactions pay the highest fee rates, as well as the block gas limit. The gas limit sets the maximum amount of processing and storage that can be done per block [Dan17, 60].

4.8 Transactions

An Ethereum transaction refers to an action initiated by an externally-owned account, in other words an account managed by a human, not a contract. For example, if Bob sends Alice 1 ETH, Bob's account must be debited and Alice's must be credited. This state-changing action takes place within a transaction. Transactions come from external accounts, which are usually controlled by human users. It's a mechanism for an external account to provide the EVM instructions to do a task. To put it another way, it's a method for an external account to send a message to the system. A transaction in the EVM is a cryptographically signed data package storing a message, which tells the EVM to transfer ether, create a new contract, trigger an existing one, or perform some calculation. Contract addresses, like users with external accounts, can be transaction recipients [Dan17, 60]. Transactions that affect the EVM's state must be broadcast to the

Figure 4.1: Illustration of an Ethereum transaction `https://ethereum.org/en/developers/docs/transactions/`

entire network. A miner will execute the transaction and propagate the resulting state change to the rest of the network once any node broadcasts a request for a transaction to be conducted on the EVM[12]. Transactions, which do not alter the global state are free of charged and are referred to as Calls. In the source code, as seen in the source code in 5.1, *getPoolHash(uint32 poolId)* is marked with the keyword view, which means the function itself does not alter the state of the network. Contrary to the function *setPoolHash(uin32 poolId, bytes32 poolHash)* which does alter the networks state and therefore generates cost for the fixity storage. In this thesis, writing actions on the blockchain are often referred to as transactions; and reading actions are referred to as calls. Table 4.8 presents

[12] `https://ethereum.org/en/developers/docs/transactions/`

the gas cost of the most relevant network operations used in this thesis according to the Ethereum yellow paper [Woo14, 27].

Operation Name	Gas Cost	Description
codedeposit	200	gas cost per byte the compiled smart contract
txcreate	32000	create a new smart contract
transaction	21000	retrieve the current balance of an account
txdatazero	4	gas cost for every zero byte of the transaction data
txdatanonzero	16	gas cost for every non-zero byte of the transaction data
sset	20000	set a persistent variable for the first time in the contract
sreset	2900	the cost for updating a persistent variable

4.9 Wrap Up

The learnings in this Chapter is on how to interact with the Ethereum network, which tools to use and how the computational cost comes about. I have presented that it is important to prune all unnecessary transaction and storage data to mitigate the ever-increasing chain size of the Ethereum blockchain and reduce the cost for the operator of the fixity storage presented in Chapter **??**.

CHAPTER 5

Fixity Storage

5.1 Fixity Information

Fixity Information provides the Data integrity checks or validation/verification keys used to ensure that the particular Content Information object has not been altered in an undocumented manner. Fixity Information includes special encoding and error detection schemes that are specific to instances of Content Objects, it does not include the integrity preserving mechanisms provided by the OAIS underlying services, error protection supplied by the media and device drivers used by Archival Storage. The Fixity Information may specify minimum quality of service requirements for these mechanisms [fSDS12, 4-30]. With fixity information you are able to answer questions regarding the authenticity and integrity of an object. *Have you received the files you expected?* When fixity information is provided with objects upfront, it can be used to validate that you have received what was intended for the collection. *Is the data corrupted or altered from what you expected?* Once you have generated baseline fixity information for files or objects, comparing that information with future fixity check information will tell you if a file has changed or been corrupted. *Can you prove you have the data/files you expected, and they are not corrupt or altered?* By providing fixity information alongside content, you enable your users to verify that what they have is identical to what you say it should be. This supports assertions about the authenticity and trustworthiness of digital objects [KK$^+$17, 3]. Fixity is a key concept for the long-term preservation and management of digital material for many reasons. Previous scholarship on fixity has shown its vital importance in discovering changes to data and all that this error-checking can imply: authenticity and renderability of files, trustworthiness of institutions, and system monitoring/maintenance. Despite the centrality of fixity to the field of digital preservation, there is little prescriptive guidance on when and how to create fixity information, where to store it, and how often to check it. This absence is not without reason, however: the incredible variety of organizational structures, priorities, staffing

levels, funding, resources, and size of collections held by institutions that do digital preservation make it difficult to establish a single set of one-size-fits-all best practices [KK+17, 38].

5.1.1 Generating Fixity Information on Ingest

It is important to check the fixity of content transferred to you when you bring it under your stewardship. Whenever possible, it is ideal to encourage content providers or producers to submit fixity information along with content objects. You can only provide assurance about the fixity of content overtime once you have initial fixity values, thus it is imperative to document fixity information as soon as possible. If fixity information is not provided as part of the transfer, you should create fixity information once you have received the materials [KKG+14, 4].

5.1.2 Fixity Checks

In addition to checking fixity before and after transfer, collections of digital files and objects should be checked on a regular basis. There are a range of systems and approaches focused on checking the fixity values of all objects at regular intervals. This could be monthly, quarterly, or yearly for example. The more often you check, the more likely you are to detect and repair errors [KKG+14, 4]. The information must be put to use, in the form of scheduled audits of the objects against the fixity information. Additionally, replacement or repair processes must be in place. Ideally these will have been tested before being needed. All of this is critical for bit-level preservation, but ensuring fixity does not mean that the object is or will be understandable. Long term access is also contingent on ones ability to make sense of and use the contents of the file in the future [KKG+14, 2]. Most fixity procedures involve a computational method that takes a digital file as input and outputs an alphanumeric value; this output value is used as a baseline comparison each time the fixity check is rerun [KK+17, 5]. For example, fixity checks may occur at different times depending on the institution's environment: during initial deposit only; during any file transmission; during scheduled backup routines; or periodically at specified times or when manually triggered [KK+17, 7]. *Throughput:* The rate of fixity checking is going to be dependent on how quickly you can run the checks, the complexity of your chosen fixity instrument, and how much of your resources (e.g., CPU, memory, bandwidth) can be used for this operation. This can become a choke point as the amount of digital content increases but the infrastructure to perform the checks stays the same. In a situation like this, the fixity checking activities can adversely affect other important functions like delivery of the content to users.

5.1.3 Fixity Instruments

For a given fixity instrument, see Table 5.1, the harder it is to find two objects that result in the same fixity information, the more "collision resistant" that instrument is. This is important mostly for preventing the concealment of intentional changes to objects. For

Table 5.1: Various Fixity Instruments [KK+17, 6]

Fixity Instrument	Definition	Level of Effort and Return of Investment
Expected File Size	File size that differs from the expected can be an indicator of problems, for example by highlighting zero byte files	Low level of effort and low level detail. File size is auto-generated technical metadata that can be viewed in Windows Explorer or other common tools.
Expected File Count	File count that differs from the expected can be an indicator that files are either added or dropped from the package.	Low level of effort and low level detail. File count is auto-generated technical metadata that can be viewed in Windows Explorer or other common tools.
CRC	Error detection code, typically used during network transfers.	Low level of effort and moderate level of detail. CRC function values, which are variable but typically 32 or 64 bit, are relatively easy to implement and analyze.
MD5	Cryptographic hash function	Moderate level of effort and high level of detail. CPU and processing requirements to compute the hash values are low to moderate depending on the size of the file. The output size of this hash value is the lowest of the cryptographic hash values at 128 bits.
SHA1	Cryptographic hash function	Moderate level of effort, high level of detail, and added security assurance. Due to its higher 160-bit output hash value, SHA-1 requires more relative time to compute for a given number of processing cycles CPU and processing time than MD5.
SHA256	More secure cryptographic hash function	High level of effort, very high level of detail, and added security assurance. With an output hash value of 256 bits, SHA-256 requires more relative time to compute for a given number of processing cycles CPU and processing time than SHA-1.

example, expected file size and expected file count are extremely vulnerable to collision: it is very easy for someone to replace an object with one that matches in file size. It is also possible (although unlikely) for an unintentional change (such as corruption or human error) to result in an object with the same fixity information for instruments that have low collision resistance. Of the fixity instruments described above, the cryptographic

hash functions (MD-5, SHA-1, and SHA-256) are the most collision resistant; SHA-256 is recommended for applications where security is important. However, performing fixity checking and replacing damaged objects is critical for any preservation system, and using any fixity instrument is much better than none at all. Note that as the level of security of the hash function increases, so do the time and resources needed to compute [KKG+14, 7].

5.1.4 Storage Medium

In object metadata records: In many cases, you will want to record some file or object fixity information wherever you store and manage the metadata records. These metadata records are actually stored as discrete files or in databases. This is particularly useful for maintaining originally submitted or generated fixity information as part of the long-term object metadata. *In databases and logs:* For checks you run at given intervals you may not want to be constantly adding to your object metadata records. In this case, it makes sense to keep running fixity information in databases and logs that you can return to when needed. *Alongside content:* Its often ideal to have fixity information right alongside the content itself. That way, if you have problems with other layers in your system, or want to transfer some set of objects, you still have a record of previous fixity values alongside your content. For example, the BagIt specification includes a requirement for a hash value for the bagged content alongside the content. Similarly, some workflows involve creating *.md5 files, which are simply text files with the md5 hash, named identically to the file it refers to, but with an additional .md5 extension. *In the files themselves:* When a checksum is for a portion of a file, it may make sense to store the information directly in the file. Note that this only makes sense when storing sub-file fixity information within a file. Adding fixity information for an entire file to the file itself changes the file and therefore changes its fixity value [KKG+14, 7]. *Ethereum Blockchain:* In this thesis I have chosen the blockchain to be the storage-medium for the fixity information. Its immutable state and availability in addition to its novel use-case in digital archives have influenced my decision.

5.2 Interface

A fixity storage must persist any kind of fixity information, in my thesis SHA256 values, for long term and guarantee that the persisted content is unaltered until retrieval of the content. I have chosen the Ethereum network as a medium for persisting file fixity information, see Section 5.1.4 for my reasoning. In Figure 5.1 the lifecycle of fixity information is presented, where at some point in time the information is ingested into the storage and after a certain time interval the information is fetched and compared to the retrieved SHA256 value from the digital object of the archive. If both cryptographic hashes match, the object is guaranteed to be unaltered.

5.3 Implementation

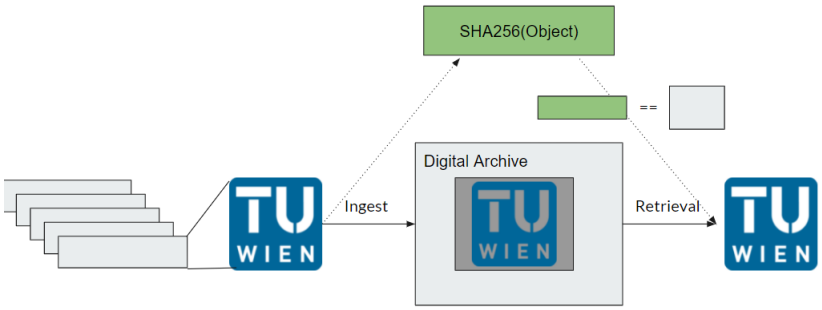
The fixity storage presented in this thesis is implemented in Solidity, a programming language designed for the EVM, see Section 4.3. The basic functionality of the smart contract, as described in Section 1.5 can be seen in the source code presented in Listing 5.1

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.4.22 <0.9.0;
3
4  contract FixityStorage {
5    mapping(uint32=>bytes32) pools;
6    address creator;
7
8    constructor()public{
9      creator = msg.sender;
10   }
11
12   function getPoolHash(uint32 poolId) public view returns(bytes32) {
13       return pools[poolId];
14   }
15
16   function setPoolHash(uint32 poolId, bytes32 poolHash) public {
17     require(msg.sender==creator);
18     pools[poolId]=poolHash;
19   }
20 }
```

Listing 5.1: MVP source code of the fixity storage deployed on the Ropsten test network https://Ropsten.etherscan.io/address/0x0243c7aa552730E8C6F7ED25A480a7C0c88a70f0

where *getPoolHash(uint32 poolId)* implements a read function; *setPoolHash(uint32 poolId,bytes32 poolHash)* implements create and update function. The mapping type is

Figure 5.1: Example of a fixity storage

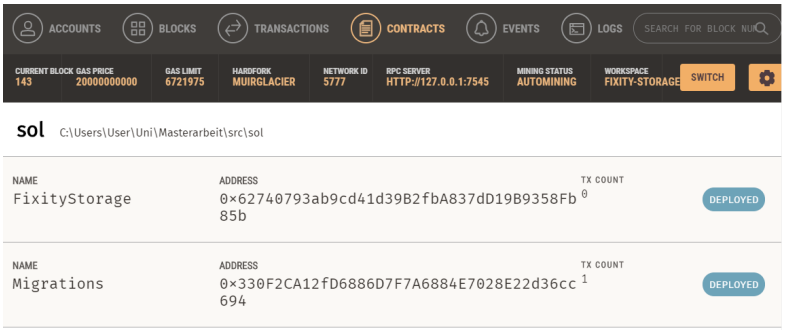


native in solidity which implements a hash map consisting of a key and a value, where in this case the key is an integer representing the *poolId* and the value is a *bytes32* object representing the SHA256 root hash of the pool. The *poolId* is the reference to the local pool in the digital archive, with which fixity information can be retrieved for a certain pool from the contract. The solidity language presents a convenient method to prevent unauthorized calls to the setPoolHash() method, which is *require(msg.sender==creator)*. The native method require is a "guard" function which improves the readability of the smart contract code which fires a REVERT instruction if the condition is not met. The condition in this case is, that only the creator, which is set in the constructor, of the fixity storage is able to create and alter the information stored on the blockchain.

5.4 Deployment

I utilized truffle[1] to run my deployment of the smart contract, it brings built-in smart contract compilation, linking, deployment and binary management with automated contract testing. The reasoning behind my decision is that, truffle has all the tools needed to implement a smart contract in one package and therefore reduced complexity in the development process. In the first iteration of the development process, I used the graphical user-interface Ganache[2] to get a better feeling for the Ethereum blockchain, see Figure 5.2. It allows you to click through your smart contract and look at the state

Figure 5.2: Ganache, an interactive user interface for the Ethereum blockchain.



variables or functions to validate that your smart contract was successfully deployed. Ganache has a massive disadvantage when doing high throughput computing, it seems that it is not suited to withstand 10000 transactions built in a python for loop. Therefore, I only used the GUI it in the beginning of the experiment where I only persisted about 10 objects at a time without problems. For high throughput computing, truffle offers a command line tool to interact with the blockchain. The command line tool was resistant and showed no weakness when persisting 10000 objects. Truffle also allows to config other networks, e.g. the Ropsten test network, which can be defined as a parameter in the

[1] https://trufflesuite.com/index.html
[2] https://trufflesuite.com/ganache/index.html

deployment process. The deployment requires to have some ETH token in your account to pay the miner to integrate your smart contract in a block. Truffle offers some utility regarding automated deployment, which are migrations. Migrations are JavaScript files, which are responsible for staging the deployment tasks and running deployment scripts. I used the migrations feature in order to interact with various Ethereum networks, in my case the Ropsten test network and a local test environment. The migrations feature requires to have a smart contract deployed on the blockchain, which causes additional cost (191943 gas) to the fixity storage, therefore I have decided not to use the migrations feature, as it is controversial in the community and seen as unnecessary traffic and cost[3]. The deployment cost of the smart contract can be calculated as follows:

$$\begin{aligned} C_{deployment} &= transaction + txcreate + codedeposit + txdatanonzero + txdatazero \\ &= 21000 + 32000 + 200 * 832 + 226 * 4 + 800 * 16 \\ &= 233104 \end{aligned} \tag{5.1}$$

where *transaction* is the base cost for a transaction; *txcreate* is the operation used to create a smart contract; *codedeposit* is the gas cost for each byte of the runtime bytecode of the smart contract, which is 832 bytes and can be seen in the Input Data field of the transaction on Etherscan[4] $G_{txdatanonzero}$ is 16 gas for each non-zero byte in the compiled bytecode of a transaction;$G_{txdatazero}$ is 4 gas for each non-zero byte in the compiled bytecode of a transaction; and $Contract_{bytesize}$ is the size of the compiled bytecode of the contract, see Section 4.7 for the exact gas amount for each transaction. The amount of gas consumed by the deployment of the decentralized fixity storage is 166079 gas on the Ropsten testnet, the transaction used to deploy the contract can be found on Etherscan[5],where additional information can be read. The address of the smart contract is *0x0243c7aa552730E8C6F7ED25A480a7C0c88a70f0*.

5.5 Authorized Access

In solidity one can require a certain address to access a function, the keyword requires can be used so that a transaction from an unknown address can be reverted and only the owner of the creator address can update the mapped SHA256 values in the smart contract. An interesting topic is also how the private key in the archive may be managed, which is not part of this thesis, but something like a multisignature wallet may be used to split the responsibility of the owner address in the contract. Since the entity which controls the private key of the smart contract is able to perform update operations which can lead to unwanted actions. In the worst case, an unwanted action may be ignored, since the older value is not deleted on the blockchain. Therefore, the key user management of the smart contract can be used as a next step in further research. For this thesis

[3] https://github.com/trufflesuite/truffle/issues/503
[4] https://ropsten.etherscan.io/tx/0x844f76cfff6e00f29487f6fe3c99d8a69eab576e7c190e5b392745b48924
[5] https://ropsten.etherscan.io/tx/0xf383c4bf0a5c32dd3369b02f68fd4e4400ef59343ad472bc96a28827f32c

I assume that the private key is well managed and each transaction coming from the master address is legit.

5.6 Cost of interacting

The cost of interacting with the fixity storage depends on the desired action. There are three different way to interact with the contract: (1) create a new entry (2) update an existing entry and (3) read an entry. The cost of a transaction, which invokes the setPoolHash function and ultimately stores 256 bit on the blockchain can be calculated with Equation 5.2

$$\begin{aligned} C_{setPoolHash} &= gasAmount * gasPrice * ethPrice \\ &= 42368 * 0.00000005 * 4000 \end{aligned} \tag{5.2}$$

which results in $8.47 on average for an ETH price of $4000. The *gasAmount* in Equation 5.2 can be calculated as follows:

$$\begin{aligned} gasAmount &= transaction + sset + txdatazero + txdatanonzero \\ &= 21000 + 20000 + 16 * 68 + 4 * 70 \\ &= 42368 \end{aligned} \tag{5.3}$$

where *txdatazero* is the number of zeros in the transaction data and *txdatanonzero* is the amount of non-zero bytes in the transaction data. The transaction data for Equation 5.3 can be found here in the Input Data field of the transaction on Etherscan[6]. The cost for updating an existing entry can be calculated in the same way as for creating ones, with the difference that the costly *sset* operation can be spared and therefore updates costs 20000 gas less than creates. Reading an entry is free of charge, since no state changes has to be forwarded to the blockchain. To explain the cost of the non-zero and zero bytes can be explained by looking the following Input Data

```
0x178d2929000000000000000000000000000000000000000000000000000000000000000000000000000001760d27083f6e2d1c46a65938c03a0c52dccf55cb4eb68a720e6efe3a8851f78
```

where zero and nonzero bytes are counted and multiplied by their respective gas cost, which in this example is $txdatazero + txdatanonzero = 4 * 70 + 16 * 68 = 1368$. The amount of non-zero bytes can be interpreted as the amount of effort given to the network.

5.7 Wrap Up

In this Chapter I have presented the various forms of fixity information with their respective advantages and disadvantages together with my reasoning on why I have

[6] https://ropsten.etherscan.io/tx/0x8839e03f0143bad34060fa909c35d30f2edb22dd4fdac0264de8a

decided to utilize SHA256 cryptographic hashes. I have also presented various storage-media for fixity information and why it is so important that these storage guarantee for immutability in regard to history forgery. The interactions with the fixity storage are tested in an experimental environment, presented in Chapter 6, where the research questions regarding the cost and efficiency will be answered.

CHAPTER 6

Experiment

6.1 Setup

The experiment written in Python in form of a Jupyter Notebook. At first the function for providing the optimal pool size was implemented, the exact implementation can range from Bernoulli experiment to another form. When the optimal pool size is found, which is the second research question where I created 10000 objects and assign them to a certain pool and assign them a corruption rate ranging from 0.01 to 0.2. The corruption rate is the base for the Bernoulli experiment. After object creation, the objects are ingested into the archive, during this process each pool is persisted on the blockchain. This happens in an interaction between a python client and a local installation of the Ethereum blockchain. Ganache is used for local development on the Ethereum side and pythons Web3 is used as a client. While ingesting and uploading onto the blockchain I will monitor the amount of gas used and the exact number of write transactions. The more exact method of monitoring the operation cost is by adding up the gas cost, since gas can be transformed into ETH. The first measurement will be theoretical and the second empirical where I monitor the amount of ETH available for the experiment account. e.g. if I start the experiment with 100 ETH and at the end I have 90 ETH left the operation cost was 10 ETH. If the experiment local is successful I will deploy the smart contract on the Ropsten testnet and add time measurement to the process since the blockchain is now real and distributed on the network. I expect the time needed to ingest and upload on the blockchain to increase the cost should stay the same. When the objects are ingested into the dummy archive and the pools are uploaded onto the blockchain I will retrieve random samples from the original set of objects and check if they are corrupt, maybe I will assign corruption at random to this random sample, and then I have to implement a repair function, this function also affect the operation cost, since when a corrupted file is found the whole pool has to be re-computed resulting in pool size + 1 local transactions and one blockchain write transaction. In the retrieval process I will monitor the time and

nullify the cost since read operations on the blockchain are free of charge, nonetheless I will count the number of transactions needed for the process. After retrieval and repairing the pools I will look into the number of transactions needed, and the time needed for the whole process. The result will be one row of a table and the experiment can be redone with different pool sizes or corruption rates to get a nice table with descriptive statistics. At last, I will split the objects and metadata and adapt the corruptions rates and redo the whole experiment and compare the results.

6.2 Object Identity

When and where should the pool ID be stored, there are possibilities that the pool ID is assigned with pool creation, but that would mean that I have to update the object which is supposed to preserve, should the pool ID be in the metadata of the object= if no, where should we store the link between the object and its respective pool. It is easy to keep the links in the local Jupyter notebook but in an industrial environment keeping those links is rather hard. For now, I keep the transaction hash with the pool object.

6.3 Object

An object in this experiment is simply a SHA256 value, since the preservation process does not care if a picture, text or video is secured by the hash. An object also holds the reference to its pool, where in a real case the poolId is stored in the object's metadata. For the sake of the experiment, the initial bulk is numbered from 1 to N, where the SHA256 value is then assigned to the object. An object also has a float value which indicates the chance of it of being corrupted, this value is used in the Bernoulli experiment to determine the optimal pool size based on the corruption rate of all objects. The final member variable of the object is a boolean flag which indicates whether an object is corrupted or not.

6.4 Pool

A pool in this experiment is a collection of objects with size k. The root hash of a pool is a hash-list of every object in the pool.

6.5 Archive Mock

For the sake of the experiment I will mock the functions of a digital archive in python with the following relevant functions implemented: (1) $bulk_ingest$ (2) retrieve (3) $get_objects_by_poolId$ (4) repair. (1) Is for ingesting a bulk of objects with their respective metadata, it is expected that the poolId is already set for the object. The bulk itself does not know anything about the pool sizes or the amount of pools. The archive therefore is independent of the implementation of a pool and only need to know the poolId of

an object. The poolId may be stored in the Preservation Description Information of an object [Lee10]. (2) Retrieve a single object from the archive. (3) The implementation of this function in a real digital archive may vary from my implementation, where I return all objects stored in the archive with matching poolId. This function is here to provide information regarding the original objects of a pool, this is necessary to rebuild the pool and recalculate the pool hash for someone who might want to check whether an object in the archive got corrupted. (4) The function which handles the case when a corrupted pool was found. It is a costly function where one write transaction to the blockchain and additional data scrubbing has to be made. When a pool is found to be corrupted, each object in the pool is suspected to be corrupted too and therefore each object has to be replaced by a copy and reassembled in a pool, which hash is then again persisted on the blockchain. The mock also provides a function to simulate corruption, where each object in the archive has a chance to corrupt itself.

6.6 Program Flow

During development, I have seen that the pool size should not be too high. When a corrupted pool is found, make sure to not double write the same pool to the blockchain You even got an advantage when you repair objects, since you have to scrub each object in the pool with a fresh copy.

6.7 Dataset

I analyzed the format-corpus[1] from Open Preserve Foundation[2] to get a better understanding of various file formats, mostly on how volatile they are. The latest commit for my analysis was *commit 4e4b9a34540f72612ba6eab2d28bccceb7a848ae*[3] on the 16th of February. The format-corpus is well-structured in a public GitHub repo with a decent amount of reputation in form of GitHub stars, where other datasets did provide corrupt links;were not available or did expect a tedious amount of time to download and re-structure them. Convenience; diversity of file extensions; and reputation of the organization affected my decision to use the format-corpus for analysis. For the analysis I used the python package folderstats[4] which transforms a directory into a pandas[5] dataframe. The repository contains 1560 files with 90 distinct file extensions, e.g. 986 *.xml* files which are mostly PRONOM[6] registry files or *pom.xml* files in case of java projects. In Figure 6.1 you can see the distribution of file extensions where *.xml* files make 62.2 percent of the portion and PDF with 6.79 percent as the second-largest portion.

[1] https://github.com/openpreserve/format-corpus

[2] http://openpreservation.org/

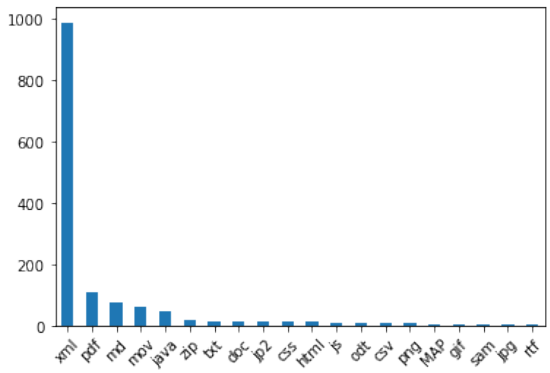
[3] https://github.com/openpreserve/format-corpus/commit/4e4b9a34540f72612ba6eab2d28bccceb7a848ae

[4] https://pypi.org/project/folderstats/

[5] https://pandas.pydata.org/

[6] https://www.nationalarchives.gov.uk/PRONOM/

Figure 6.1: Distribution of file extensions in the format-corpus of Open Preserve Foundation. https://github.com/openpreserve/format-corpus



6.7.1 Transformations

To get a baseline estimate on how volatile certain extensions are due to updates and alterations, I analyzed Git logs and count how many times a file extension was involved in a commit. The resulting column *positives* was always higher or equal than the occurrence of a file extension, since each file was involved in at least one commit, the initial commit of the file. To count the commits I used the python package gitphyton[7] and utilized the git command –log. For each file in the repository, I fired up the command *git log –oneline filename* in python which resulted in one or multiple lines of logs. Latter I used the multiline log as an input for the *.splitlines()* function which results in an array of log lines, the length of this array minus 1 (the initial commit) is used to determine the new column *positives* in the dataframe which shows if and how often a certain file has changed over the lifetime of the git repository. This method of determining the volatility of a certain file extension is by no means ideal, but there is no other method to look into how often a certain file has changed without monitoring them on a system for a certain time interval. Therefore, I have decided to use the amount of file alterations in the git repository as a rough estimate. The estimated prevalence rate p of a file extension is calculated with $positives/N$, see Table 6.1 I applied Equation 6.1, created by [REHHR20], to each row in Table 6.1 to calculate the optimal pool size for each file extension.

$$k = 1.24 * (positives/N)^{-}0.466 \tag{6.1}$$

6.8 Wrap up

In this Chapter, I have presented the environment and dataset needed in order to answer the research questions which are presented in Chapter 7 TODO weiterarbeiten

[7] https://gitpython.readthedocs.io/en/stable/

Table 6.1: Volatility of certain file extensions in the format-corpus dataset

extension	N	positives	p	k
XML	986	432	0.43	2
PDF	106	2	0.01	8
MD	74	17	0.22	3
MOV	61	0	0.00	61
JAVA	47	39	0.82	2
ZIP	17	0	0.00	17
TXT	14	2	0.14	4
DOC	13	0	0.00	13
JP2	12	0	0.00	12
HTML	11	6	0.54	2
CSS	11	4	0.36	2
JS	10	3	0.3	3

CHAPTER 7

Evaluation

The proposed fixity storage is evaluated on a local installation of the Ethereum blockchain and on the online Ropsten test network, where ETH used for transactions are free of charge. The two key parameters for evaluation are the operation cost and efficiency. *Operation cost* is calculated by multiplying the number of relevant cost transactions with the average gas cost of a writing transaction on the blockchain, see Equation 7.1

$$C = J * C_{setPoolHash} = \lceil N/k \rceil * gasAmount * gasPrice * ethPrice \tag{7.1}$$

where J is the number of pools on ingest; $C_{setPoolHash}$ is the cost for a transaction in gas, see Section 5.6. I intend to present the operation cost in the form of gas, the unit that measures the amount of computation effort required to execute specific operations, instead of the cost in USD or EUR. This is because, the amount of gas used during the experiment should stay constant, whereas the price of the ETH token fluctuates heavily. Therefore, the amount of gas is a better indicator on how costly the fixity storage is. *Efficiency* $E(S)$ is measured by the number operations needed utilizing pooled testing compared $T(S_{pooling})$ with the individual testing strategy $T(S_{individual})$, where the efficiency of pooling strategy S is expressed by Equation 7.2

$$E(S) = N/T(S) \tag{7.2}$$

where, assuming the preservation process of 10,000 digital objects without pooling requires N operations and the preservation of the same objects using strategy S requires $T(S)$ operations. In the case of individual testing the efficiency is 1, whereas if strategy S requires two times fewer operations the efficiency E(S) is 2 [ŽLG21, 4].

What is the optimal pool size based on the corruption rates of digital objects in the archive regarding cost and efficiency?

Corruption rate p represents the prevalence of corruption, e.g., when I assume that 2000 of 10000 objects will be corrupted during the preservation process, $p = 2000/10000 = 0.2$.

Expected Operation Cost: The optimal pool size, which will result in the lowest number of cost relevant transactions is the number of pools J, as seen in Equation 7.1a, because I only must write onto the blockchain during the ingest process where the root hashes of pools are persisted. After the first Iteration of the experiment, the number of corrupt objects per positive pools were included in the first draft of Equation 7.1, where I assumed that I had to re-calculate corrupt pool hashes and re-store them on the blockchain, but these "repairing" actions can be done locally through data scrubbing. I only need to know that the pool is corrupt, then I can to substitute each object in the pool with a correct copy in the archive. For this part of the research question, local operations are out of scope because they do not cause direct cost on the blockchain, therefore the optimal pool size calculated by Equation 7.1 is N, see Figure 7.1a. A pool size of N results in exactly 1 cost relevant transaction since I have combined every object on ingest into a hash-list which's root will be stored on the blockchain. This solution does not scale well, e.g., picture the process of retrieving a single object from the archive. In order to guarantee that the object is unaltered, I would have to re-compute the hash list from every object in the archive (or the bulk ingest in question). Additionally, if the single pool is corrupted, I must replace the whole bulk with copies. So, there must be an answer, which rewards smaller pool sizes to avoid too much data scrubbing. The number of data scrubbing operations is the number of objects in positive pools on retrieval and is calculated with Equation 7.3,

$$T(S) = J_+ * k = ((1 - (1 - p)^k) * \lceil N/k \rceil) * k \tag{7.3}$$

where $(1 - p)^k$ is the probability of a pool of size k being negative at a prevalence of p and the probability of a pool being positive is $1 - (1 - p)^k$. To find the optimal pool size regarding the amount of data scrubbing operations, I had to minimize Equation 7.3, as seen in Figure 7.1b. To find the optimal pool size in terms of operation cost and efficiency, operation cost and data scrubbing has to be accounted, which results in the final Equation 7.4

$$T(S) = J + J_+ * k = \lceil N/k \rceil + ((1 - (1 - p)^k) * \lceil N/k \rceil) * k \tag{7.4}$$

where J is the number of pools and $J_+ * k$ is the number of objects in corrupted pools. Therefore, the number of expected operations is the number of writing operations on ingest plus the number of data scrubbing operations on retrieval. By adding the amount of data scrubbing operations, the optimal pool size gets significantly lower, see Figure 7.1c. The optimal pool size k can be determined by finding the global minimum of Equation 7.4, which results in the highest efficiency in Equation 7.2. In the experiment, I have utilized Equation 7.5 presented by [REHHR20, 3] and round the pool size up in order to avoid non integer pool sizes.

$$k = \lceil 1.24 * p^{-}0.466 \rceil \tag{7.5}$$

In Table 7.1 and Figure 7.3 it is shown that for ingest bulks with higher prevalence of corruption smaller pool sizes are favorable. The highest efficiency $E(S)$ can be achieved when the prevalence rate is the lowest, where larger pool sizes are favorable.

Table 7.1: Expected transaction throughput with different prevalence of corruption rates

N	p	k	E(S)
10000	0.010	11	5.16
10000	0.037	6	2.77
10000	0.065	5	2.18
10000	0.121	4	1.71
10000	0.177	4	1.51
10000	0.205	3	1.45
10000	0.316	3	1.29
10000	0.372	3	1.24
10000	0.400	2	1.22

To what extent can pooled testing increase the efficiency and reduce cost for a fixity information storage service on the Ethereum blockchain?

A context-sensitive approach in the medical field was proposed in 2020 were members of homogeneous groups were pooled, e.g. families, office colleagues or neighbors, and is proven to be more effective than individual testing [DBK20, 4]. In this thesis, the idea of grouping similar files to increase efficiency of the pooling strategy is implemented by grouping various file extensions, estimating their prevalence in form of alterations and calculating the efficiency per file extension as presented in Equation 7.2. The dataset used is presented in Section 6.7. In Figure **??**, it is shown that the volatility in the repository only affects a few file extensions, where 77 out of 90 file extension where never altered and have therefore a p value of 0.00. These unaltered file extensions are grouped with a group size of $Group_N$, which results in large bulks in the ingest process that have no impact on the retrieval since I assume that no object in this particular group will be altered during the preservation process. For instance, in Table 7.2, the file extension MOV has an prevalence rate of 0.00, and therefore I can assign a group size of N to reach the maximum overall efficiency.

To calculate the efficiency of the two stage hierarchical pooling strategy, I took the mean p value of the data in Table 7.2 as an estimator for the overall alteration rate which resulted in 0.06. The input for Equation 7.4 was therefore N=1560; p=0.06 and k=5.0. With context-sensitive pooling, another increase in efficiency can be made. In Table 7.3 I compare the two strategies presented before, with respect to Equation 7.2, where the Two-Stage-hierarchical pooling algorithm has achieved an efficiency rate of 2.24 and context-sensitive pooling has achieved an efficiency rate of 3.03.

RQ 3 Given that metadata has a higher corruption rate, what effect has the split of metadata and objects on the operation cost?

To know what effect the split off metadata has, I had to double the initial 1560 file extensions and assign the postfix .meta to the newly created rows resulting in Table 7.4, where $p = 0.005$ for each file extension and $p_{meta} = 0.800$. I have calculated the efficiency

for context-sensitive and two-stage hierarchical for various combinations of p and p_{meta} and the result is, that with split off metadata the context-sensitive strategy is much worse than the two-stage hierarchical algorithm. For instance in Table 7.5, the context-sensitive strategy never even reaches a 1.00 efficiency, which would state that it performs worse as the individual testing in terms of efficiency, whereas the two-stage hierarchical strategy performs better with split off metadata when the p value is low enough. This is because I can assign larger group sizes when the majority of file extensions (the objects) in the dataset do have a very low prevalence rate p. This is due to the fact, that the average p value is calculated after grouping the file extensions, where we have only one file extension where the p value is very vast, the metadata.

Table 7.2: In group efficiency of various file extensions

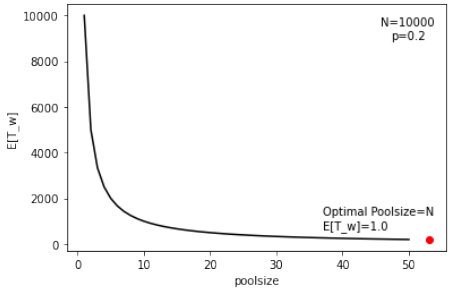
extension	N	p	k	T(S)	T($S_{writing}$)	T($S_{scrubbing}$)
XML	986	0.43	2	1.21	2.00	0.64
PDF	106	0.01	8	3.60	7.57	0.12
MD	74	0.22	3	2.96	2.00	0.41
MOV	61	0.00	61	61.00	2.00	1.00
JAVA	47	0.82	2	1.21	1.95	0.82
ZIP	17	0.00	17	17.00	17.00	1.00
TXT	14	0.14	4	1.33	3.50	0.25
DOC	13	0.00	13	13.00	13.00	1.00
JP2	12	0.00	12	12.00	12.00	1.00
CSS	11	0.36	2	1.07	1.83	0.50
HTML	11	0.54	2	1.06	1.83	0.60
JS	10	0.3	1.08	2.50	0.50	

Table 7.3: Efficiency of context-sensitive pooling vs. two stage hierarchical pooling

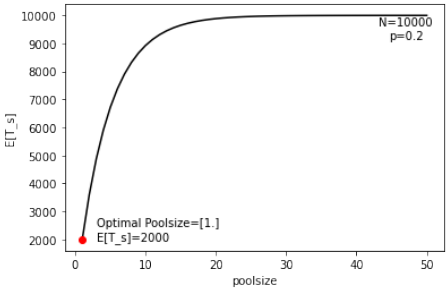
Strategy	T(S)
Individual	1.00
Two stage hierarchical	2.24
Context Sensitive	3.03

Table 7.4: Arbitrary prevalence rates p and p_{meta} in the format-corpus dataset

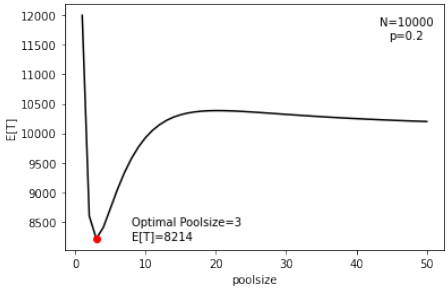
Extension	N	p
METADATA	1560	0.990
XML	986	0.001
PDF	986	0.001
MD	986	0.001
MOV	986	0.001



(a) Optimal pool size regarding the writing transactions. $E[T_w]$



(b) Optimal pool size regarding the data scrubbing operations $E[T_s]$.



(c) Optimal pool size regarding the total operations $E[T] = E[T_w] + E[T_s]$.

Figure 7.1: Comparison of optimal pool sizes

Figure 7.2: Optimal pool sizes k with prevalence p

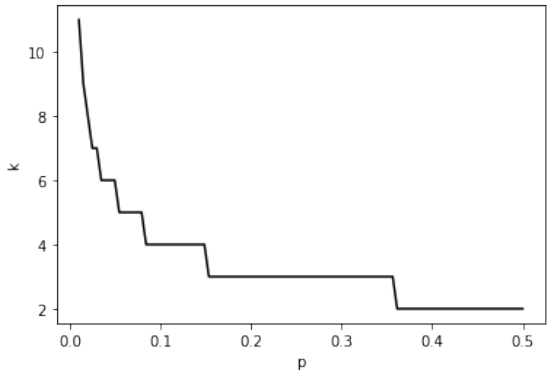


Figure 7.3: Distribution of prevalence rates p

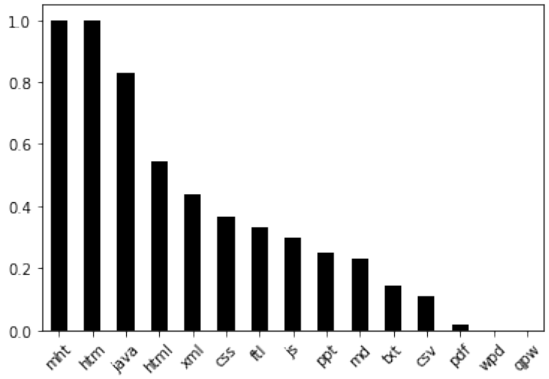


Table 7.5: Arbitrary prevalence rates p and p_{meta} in the format-corpus dataset compared with the individual testing strategy T(N) where 2044 operations are needed.

p	p_{meta}	$T(S_{two-stage})$	$T(S_{context-sensitive})$
0.001	0.99	3.10	0.76
0.002	0.98	2.95	0.75
0.006	0.97	2.72	0.73
0.023	0.90	1.86	0.68
0.042	0.82	1.53	0.66

CHAPTER 8

Discussion

The first research questions where I did research to find the optimal pool size for the fixity information had two different approaches, the first was to minimize the cost and the second was to maximize the efficiency. The two of them were inversely correlated, where to minimize the cost the pool size was actually N, since there was only one writing transaction on the blockchain when we have only one pool on ingest. On the other hand, a pool size of N is extremely inefficient on retrieval, where I have to replace the whole ingest bulk with fresh copies if only one object in the pool is corrupt, since I cannot determine anymore which object in the pool got corrupted due to the nature of hash lists. Therefore, I had to also account the number of data scrubbing operations into the optimal pool size, resulting in Equation 7.3. The pool size is therefore heavily dependent on the prevalence of corruption in the ingest bulk, where if the prevalence is high smaller pool sizes are favorable. The goal of this thesis was to reduce the cost of the fixity storage by at least 50%, this goal can be met when the optimal pool size is at least two, which is guaranteed with a prevalence lower than 35.9%, as seen in Figure 8.1. What happens if the prevalence is higher than 35.9%, the optimal pool size would be between 1.0 and 2.0 which is not practical since a pool can not include 1.42 objects, therefore I have decided to take the ceil of the result from Equation 7.5 which results in Figure 8.2. For the second research question, I have compared two different approaches for pooled testing. Where the efficiency of both strategies is compared to the efficiency of individual testing. The efficiency of individual testing for sample population of 100 objects where 20 of them are corrupt, meaning that I need 100 writing + 20 scrubbing operations during the preservation process. With two stage hierarchical pooling an efficiency of 2.24 can be achieved, meaning that this strategy is more than two times as efficient as individual testing. In the format-corpus dataset there were just a few file extensions of 90 that were altered and had a prevalence of corruption greater than 0.0, suggesting that the two-stage strategy was not optimal, since non-volatile files would give been grouped together creating unnecessary traffic on repairing a corrupt pool. Therefore, I

Figure 8.1: For a prevalence of corruption of maximum 35.9%, an optimal pool size of at least two is guaranteed

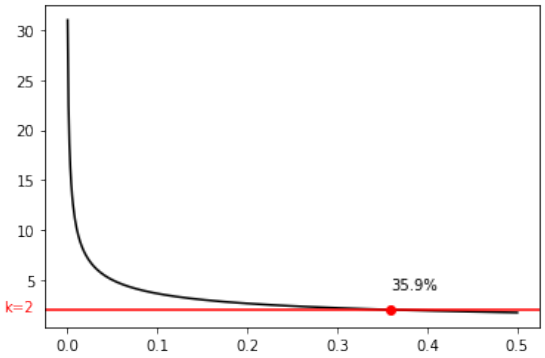
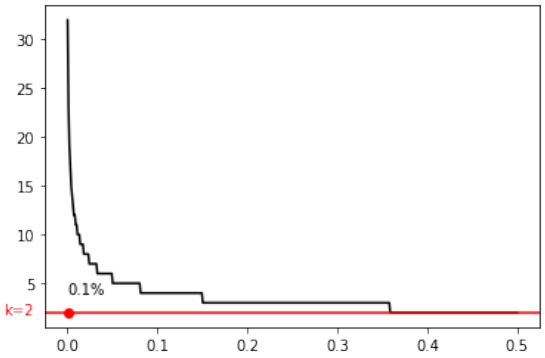


Figure 8.2: For a prevalence of corruption of maximum 35.9%, an optimal pool size of at least two is guaranteed



have proposed a context-sensitive approach where the inner group prevalence of each file extension is taken into account. Each group in the ingest-bulk is assigned to an own pool with their respective pool size. For instance, in table 7.2 the file extension PDF has 8 corrupted objects out of 106, which results in a pool size of 8, whereas the XML extension has a much higher corruption rate with an optimal poolside of 2. The context-sensitive approach takes these differences into account and pools homogenous groups, where groups with 0.0 prevalence of corruption can be grouped with a pool size as large as the inner group. With this method, groups with low prevalence can be pooled with large poolside and therefore reduce the cost and increase the efficiency even more than individual testing or two-stage hierarchical pooling. The third research question had a surprising result, given that metadata has a much higher prevalence of corruption rate than the object themselves, two-stage hierarchical pooling increases its efficiency with split off metadata whereas the context-sensitive strategy looses efficiency by a large margin. This is due, the double amount of objects needed to be processed during the preservation process where the context-sensitive approach can not make up for it, whereas the two-stage hierarchical approach is able to increase the efficiency: due to the low

average prevalence of corruption. Because I have N/2 objects with very high prevalence but with the same file extension, therefore we have only one inner group in the dataset with e.g. p=0.99 where the N-1 file extensions have a very low prevalence rate. The imbalance of alteration rates, see Table 7.3, is favorable because the average prevalence rate gets very low which results in a large pool size in the two-stage approach. The limitations of the strategies proposed in this thesis is the missing data on prevalence rates on digital objects, this is due to the fact that I would have had to monitor an archive and measure the amount of times an object has changed. In this thesis, I have arbitrarily chosen prevalence rates for the objects and their metadata, see Table 7.5. Future work may propose a method on how to estimate the prevalence rate of an ingest-bulk in order to calculate the real optimal pool size for the bulk. The results indicate that the cost for the decentralized fixity storage can be reduced, depending on the prevalence rate of the digital objects by at least 50%, given that the optimal pool size for an ingest-bulk is at least 2.

CHAPTER 9

Conclusion

Answer to research questions The optimal pool size can be calculated with two points in mind, when only regarding the cost you are able to increase the pool size until you have only one pool to ingest, whereas if you have the efficiency in mind you have to utilize smaller pool sizes utilize to minimize data-srcubbing actions on retrieval. I have presented two pooling strategies, two-hierarchical and context-sensitive pooling, which each of them is able to reduce the cost and increase efficiency. With pooled testing, I am guaranteed to decrease the cost of a decentralized fixity storage by at least 50%, see Table 7.2 column T($S_{writing}$). On the format-corpus dataset with prevalence rates estimated from git commits, the experiment with split-off metadata showed that, two-stage hierarchical pooled can decrease the cost even further whereas the context-sensitive approach shows weakness to the double amount of high prevalence groups such as the metadata, as shown in Table 7.3. The research showed that varying price for tokens on the blockchain may be neglected from calculations, since they fluctuate too much. The focus was to find the absolute numbers of computational effort in form of gas, see Section 4.7 for preserving fixity information, from which the value in EUR; ETH or other currencies may be derived at a later point. Future work may include the estimation of prevalence of corruption of ingest-bulks, my suggestion is to monitor a digital archive for a certain amount of time and count the amount of times a certain object has changed. My contribution to gather the exact computational effort, in form of gas, needed to operate a decentralized fixity information storage on the Ethereum blockchain. Additionally, I have presented a strategy to decrease the amount of operations and therefore the amount of cost relevant transactions needed to operate such an application. With pooled testing, currently utilized in the COVID-19 pandemic, the amount of cost relevant transaction has been reduced by, depending on the prevalence of corruption, by at least 50%.

Bibliography

[B+13] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[CBB+18] John Collomosse, Tu Bui, Alan Brown, John Sheridan, Alex Green, Mark Bell, Jamie Fawcett, Jez Higgins, and Olivier Thereaux. Archangel: Trusted archives of digital public documents. In *Proceedings of the ACM Symposium on Document Engineering 2018*, pages 1–4, 2018.

[CGWK20] Alhaji Cherif, Nadja Grobe, Xiaoling Wang, and Peter Kotanko. Simulation of pool testing to identify patients with coronavirus disease 2019 under conditions of limited test availability. *JAMA NETWORK OPEN*, 3(6):e2013075–e2013075, 2020.

[Dan17] Chris Dannen. *Introducing Ethereum and solidity*, volume 1. Springer, 2017.

[DBK20] Andreas Deckert, Till Bärnighausen, and Nicholas NA Kyei. Simulation of pooled-sample analysis strategies for covid-19 mass testing. *Bulletin of the World Health Organization*, 98(9):590, 2020.

[DFG+14] Paula De Stefano, Carl Fleischhauer, Andrea Goethals, Michael Kjörling, Nick Krabbenhoeft, Chris Lacinak, Jane Mandelbaum, Kevin McCarthy, Kate Murray, Vivek Navale, and Others. Checking Your Digital Content: What is Fixity, and When Should I be Checking It? *National Digital Stewardship Alliance*, 2014.

[Dor43] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

[Dur09] Luciana Duranti. From digital diplomatics to digital records forensics. *Archivaria*, pages 39–66, 2009.

[fSDS12] Consulative Committee for Space Data Systems. *Reference Model for an Open Archival Information System (OAIS)*. 2012.

[Hev07] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[HS05] Mark Hofer and Kathleen Owings Swan. Digital image manipulation: A compelling means to engage students in discussion of point of view and perspective. *Contemporary Issues in Technology and Teacher Education*, 5(3):290–299, 2005.

[HTBM17] Peijie Hou, Joshua M Tebbs, Christopher R Bilder, and Christopher S McMahan. Hierarchical group testing for multiple infections. *Biometrics*, 73(2):656–665, 2017.

[KK+17] Kim, Katherine, et al. Fixity survey report: An ndsa report. 2017.

[KKG+14] Kim, Katherine, Wayne Graham, Digital S Alliance, Aliya Reich, and Carol Kussmann. What is Fixity, and When Should I be Checking It. 2014.

[KORD10] Matthew Kirschenbaum, Richard Ovenden, Gabriela Redwine, and Rachel Donahue. Digital forensics and born-digital content in cultural heritage collections. 2010.

[Lee10] Christopher A Lee. Open archival information system (OAIS) reference model. *Encyclopedia of library and information Sciences*, 3, 2010.

[NEG+21] Roch A Nianogo, I Obi Emeruwa, Prabhu Gounder, Vladimir Manuel, Nathaniel W Anderson, Tony Kuo, Moira Inkelas, and Onyebuchi A Arah. Optimal uses of pooled testing for covid-19 incorporating imperfect test performance and pool dilution effect: An application to congregate settings in Los Angeles county. *Journal of medical virology*, 93(9):5396–5404, 2021.

[REHHR20] Francesca Regen, Neriman Eren, Isabella Heuser, and Julian Hellmann-Regen. A simple approach to optimum pool size for pooled sars-cov-2 testing. *International Journal of Infectious Diseases*, 100:324–326, 2020.

[SBP+20] Marten Sigwart, Michael Borkowski, Marco Peise, Stefan Schulte, and Stefan Tai. A secure and extensible blockchain-based data provenance framework for the Internet of Things. *Personal and Ubiquitous Computing*, 2020.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger berlin version. 2014.

[WY21] Hosung Wang and Dongmin Yang. Research and development of blockchain recordkeeping at the National Archives of Korea. *Computers*, 10(8):90, 2021.

[ZCL+20] Quanyu Zhao, Siyi Chen, Zheli Liu, Thar Baker, and Yuan Zhang. Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems. *Information Processing & Management*, 57(6):102355, 2020.

[ŽLG21] Julius Žilinskas, Algirdas Lančinskas, and Mario R Guarracino. Pooled testing with replication as a mass testing strategy for the covid-19 pandemics. *Scientific Reports*, 11(1):1–7, 2021.

Enter your text here.