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, 1. Jänner 2001

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, 1st January, 2001

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, 1. Jänner 2001

Michael Etschbacher

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

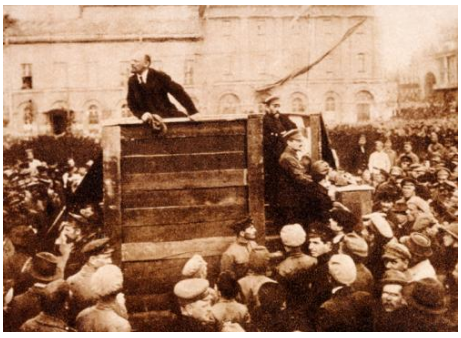
Enter your text here.

Contents

CHAPTER 1

Introduction

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 **??**, 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 **??**, using paint, razors and airbrushes. Soviet photo artists altered the historical record by literally removing Trotsky from the pictures [**?**, 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 [**?**, 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 [**?**, 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. 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 [**?**, 35].
Fixity information is usually stored in databases; object metadata records or alongside content, whereas this thesis deals with blockchain as a storage medium. [**?**] and [**?**] have shown that the Ethereum blockchain, implemented by [**?**], can indeed be utilized to ensure data integrity, but there is a problem with that: the operation cost. The cost of storing a SHA256 bit word on the Ethereum blockchain is 20.000 gas, which oscillate at the time of writing at about 5$, which means the operation cost for an ingest of 10.000 objects costs about 50.000$.
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 object hashes 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. [**?**, 1]

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

CHAPTER 2

Digital Archives

2.1 Historical

Since the creation of diplomatics we humans have archived documents and relevant pieces of history in archives.

2.2 Data Stewards

Data stewards have to prove that they handle their archive dpieces with care and integrity

2.3 OAIS

2.4 History Forgery

CHAPTER 3

Data Integrity and Authenticity

3.1 Forms of Fixity

3.2 Fixity Checks

3.3 Security of Smart Contracts

3.4 Hashing

Generally speaking, the purpose of hash functions, in the context of a blockchain, is to compare large datasets quickly and evaluate whether their contents are similar. A one way algorithm processes the entire blocks transactions into 32 bytes of data—a hash, or string, of letters and numbers that contains no discernible information about the transactions within. The hash creates an unmistakable signature for a block, allowing the next block to build on top of it [**?**, 55].

CHAPTER 4

Ethereum Blockchain

Open source blockchain networks such as Ethereum and Bitcoin are kits that allow you to set up an economic system in software, complete with account management, a native unit of exchange to pass between account. These native units of exchange are called coins, tokens, or cryptocurrencies, but the are no different from tokens in any other system: they are a form of money that is usable only within that system to simply pay your peers or to 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.py (used in this thesis)[1] to connect an applications front end (the GUI you see in a browser), via JavaScript APIs, to its back end (the blockchain) [**?**, 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 favoritable in the case of an fixity storage as presented in **??** since anyone will be able to confirm if an object has changed over time. Lets say an digital archive releases multiple objects to the public and their respective fixity information on the blockchain, citizens could then validate if the object in question is really what it is meant to be. Trust in the archive is therefore not needed anymore, which will strenghten the general trust of the archive. A block is a unit of time that encompasses a certain number of transactions. Inside that period, transaction data is recorded; when the unit of time elapses, the next block begins. The blockchain represents the history of state changes within the network database of the EVM [**?**, 43]. Transactions and state changes in the Ethereum network are segmented into blocks, and then hashed. Each block is verified and validated before the next canonical block can be placed on top of it. In this way, nodes on the network do not need to individually evaluate the trustworthiness of every single block in the history of the Ethereum network, simply to compute the present balances of the accounts on the network. They merely verify that its "parent block" is the most recent canonical block. They do this quickly by looking to see that the new block contains the correct hash of its parents transactions and state.

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

All the blocks strung together, and including the genesis block, an honorific describing the first block the network mined after coming online, are called the blockchain. In some circles, you will hear the blockchain referred to as a distributed ledger or distributed ledger technology (DLT). Ledger is an accurate description, as the chain contains every transaction in the history of the network, making it effectively a giant, balanced book of accounts [**?**, 55].

4.1 Ethereum Virtual Machine

The EVMs physical instantiation can not be described in the same way that one might point to a cloud or an ocean wave, but it does exist as one single entity maintained by thousands of connected computers running an Ethereum client [2]. A virtual machine (VM), in the Ethereum context, is one giant global computer composed of constituent nodes, which are themselves computers too. Generally speaking, a virtual machine is an emulation of a computer system by another computer system. These emulations are based on the same computer architectures as the target of their emulation, but they are usually reproducing that architecture on different hardware than it may have been intended for. Virtual machines can be created with hardware, software, or both. In the case of Ethereum, it is both. Rather than securely network thousands of discrete machines, Ethereum takes the approach of securely operating one very large state machine that can encompass the whole Earth [**?**, 48]. The EVM can run arbitrary computer programs written in the Solidity[3] language. These programs, given a particular input, will always produce the output the same way, with the same underlying state changes. This makes Solidity programs fully deterministic and guaranteed to execute, provided youve paid enough for the transaction. Solidity programs are capable of expressing all tasks accomplishable by computers, making them theoretically Turing complete. That means that the entire distributed network, every node, performs every program executed on the platform [**?**, 50]. From the perspective of a software developer, the EVM is also a runtime environment for small programs that can be executed by the network. The EVM has its own language, the EVM bytecode, to which your smart contracts compile. Solidity, which is a high-level language, is compiled into bytecode and uploaded onto the Ethereum blockchain by using a client application such as geth[4] [**?**, 51].

4.2 Smart Contracts

Smart contracts are the building blocks of decentralized applications running on the EVM, they are like the concept of classes in conventional object-oriented programming. When developers speak of writing smart contracts, they are typically referring to the practice of writing code in the Solidity language to be executed on the Ethereum network.

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

When the code is executed, units of value may be transferred as easily as data [**?**, 10]. In my thesis, the fixity storage is often referenced as decentralized application (DAPP) which is built with several smart contracts. The main smart contract used in this thesis is further described in Section **??**.

4.3 Persistence

Unlike a centralized server operated by a single company or organization, decentralized storage systems consist of a peer-to-peer network of user-operators who hold a portion of the overall data, creating a resilient file storage sharing system. These can be in a blockchain-based application or any peer-to-peer-based network. Ethereum itself can be used as a decentralized storage system, and it is when it comes to code storage in all the smart contracts. However, when it comes to large amounts of data, that isn't what Ethereum was designed for. The chain is steadily growing, but at the time of writing, the Ethereum chain is 602.95 GB for Mar 19 2022[5], and every node on the network needs to be able to store all of the data. 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 [6]. My proposed solution using pooled testing, presented in Section **??**, counters the ever increasing chain size of the Ethereum network by reducing the amount of write transaction by at least 50 percent. Each writing transaction performed from the fixity storage stores exactly one sha256 result, on the blockchain. The gas cost of storing 256 bit on the blockchain can be read from Table **??**. All transactions in Ethereum are stored on the blockchain, a canonical history of state changes stored on every single Ethereum node [**?**, 12]. Like all databases, a blockchain has a schema: rules define, constrain, and enforce relationships between entities. Motivations to break or alter these relationships can be found across industries, leading to bribery and corruption, and making blockchains trustless qualities even more attractive to business than prior generations of software and networking. In all databases, shared read/write access creates enormous complexity. Machines all over the world may experience varying latency, depending on where the database is physically located, leading to some write operations arriving out of order. This gets even more difficult if several parties are supposed to equally share a database [**?**, 20]. The nodes go through the block they are process and run any code enclosed within the transactions. Each node does this independently; it is not only highly parallelized, but highly redundant. Despite the high redundany, this is an efficient way to balance a global ledger in a trustworthy way [**?**, 50].

4.3.1 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 longterm. For a piece of data to persist

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

forever, Ethereum needs to use a persistence mechanism. For example, on Ethereum, the persistence mechanism is that the whole chain needs to be accounted for when running a node. New pieces of data get tacked onto the end of the chain, and it continues to grow - requiring every node to replicate all the embedded data. This is known as blockchain-based persistence. The issue with blockchain-based persistence is that the chain could get far too big to upkeep and store all the data feasibly (e.g. many sources estimate the Internet to require over 40 Zetabytes of storage capacity). The blockchain must also have some type of incentive structure. For blockchain-based persistence, there is a payment made to the miner. When the data is added to the chain, the nodes are paid to add the data on [7].

4.3.2 Contract-based persistence

Contract-based persistence has the intuition that data cannot be replicated by every node and stored forever, and instead must be upkept with contract agreements. These are agreements made with multiple nodes that have promised to hold a piece of data for a period of time. They must be refunded or renewed whenever they run out to keep the data persisted. In most cases, instead of storing all data on-chain, the hash of where the data is located on a chain gets stored. This way, the entire chain doesn't need to scale to keep all of the data [8].

4.4 Test Networks

. These networks are not really free of charge, but the ether token on the respective network is free to get. Although the queue of the free ether 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 recieve one ether token for the ropsten network. Other faucets, a distributer of free ether test tokens, require social network accounts to verifiy 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 ether token. The faucet request are almost all time gated, meaning that you could request ether 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 ether tokens. The one per day paradigm gets problemeatic for the experiment for the 10.000 objects, which costs about 20 ether tokens, which means I would have to request tokens fors 20 days straight to perform one experiment on a real live environment such as the ropsten testnetwork. There are multiple options to choose from for a testnetwork, from the most prominent networks presented at the official ethereum documentation https://ethereum.org/en/developers/docs/networks/ 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 this experiment

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

4.5 Costs

Miners are paid this ether 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 ether, 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 [**?**, 12]. Mining achieves the consensus required to make valid state changes, and the miners are paid for contributing to the consensus building. This is how ether and bitcoin are created [**?**, 57]. For every instruction the EVM executes, there must be a cost associated, to ensure the system isnt jammed up by useless spam contracts. Every time an instruction executes, an internal counter keeps track of the fees incurred, which are charged to the user. Each time the user initiates a transaction, that users wallet reserves a small portion to pay these fees [**?**, 58]. The fees are the driving factor of my thesis where I try to make as less transactions as possible to run a fixity storage system on the Ethereum network. The fees are dependent on the gas cost of a transaction, gas is a unit of work; it is not a subcurrency, and you can not hold or hoard it. It simply measures how much effort each step of a transaction will be, in computational terms. To be able to pay for gas costs, you simply need to add ether to your account. You do not have to acquire it separately; there is no gas token. Every operation possible on the EVM has an associated gas cost. Gas costs ensure that computation time on the network is appropriately priced [**?**, 59]. If you send a computationally difficult set of instructions to the EVM, the only person this hurts is you. The work will spend your ether, and stop when the ether you allocated to the transaction runs out. It has no effect on anyone elses transactions. There is no way to jam up the EVM without paying a lot, in the form of transaction fees, to do it. Scaling is handled in a de facto way through the gas fee system. Miners are free to choose the transactions that pay the highest fee rates, and can also choose the block gas limit collectively. The gas limit determines how much computation can happen (and how much storage can be allocated) per block [**?**, 60].

4.6 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, which change the state of the EVM, need to be broadcast to the whole network. Any node can broadcast a request for a transaction to be executed on the EVM; after this happens, a miner will execute the transaction and propagate the resulting state change to the rest of the network. [9].

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

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


```
{
  from: "0xAF8725604990d46042A50EfD9e2cB118141Bb140",
  to: "0x2C3c7752cE837bB97D6C31B4f883EAAA92BC3Ce5",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "42",
  value: "10000000000"
}
```

Transactions come from external accounts, which are usually controlled by human users. It is a way for an external account to submit instructions to the EVM to perform some operation. In other words, it is a way for an external account to get a message into 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 can be the recipients of transactions, just like users with external accounts [**?**, 60]. Table **??** presents the gas cost of the most used operation used in this thesis according to the Ethereum yellow paper [**?**, 27].

Operation Name	Gas Cost	Description
SSTORE	20000	save a 256 bit word to storage
CREATE	32000	create a new smart contract
BALANCE	400	retrieve the current balance of an account

aer

CHAPTER 5

Fixity Storage

5.1 General

Fixity can be stored in multiple places but the keypoint of it is to have some kind of reference to the object and a cryptographic hash value with it to control if the object has changed at a certain time interval. Another keypoint o fixity storages is that it should not be vulnerable to unauthorized access.

5.2 Solidity

In solidity there exists a mapping type, which is exactly what is needed for the fixity storage. The mapping type is basically a hash map consisting of a key => value, where in our case the key is a integer representing the pool id and the vlue is a bytes32 object representing the sha256 value of the pool. Since pretty much everything needed is available native in the solidity language the smart contract of the fixity storage is rather sparse. The smart ccontract has one mapping object with a getter and setter to access the map. The cost for deploying the smart contract can be dreived from the used variables in solidity.

5.3 Cost

The exapt computation o deployment cost is as follows: todo For example the upload process can be handledat time of the day where there is less trafic on the ethereum blockchain, which will result in overall less operation cost in Ether tokens and Dollar respectively. A cron expression could be used client sided to upload the pools on the least used days of the week. See figure **??**. The deployment cost of the smart contract is also dependent on the variable types used to persist data.

5.4 Access

In solidity one can require a certrain address to acess a function, the keyword requires can be used so that a transaction from a unknown address can be reverted and only the archive itself can update the mapping vlue in the smart contract. An interestng topic is also how the private key in the archive is maanged, which is not part of this thesis but something like a multisignature wallet may be used to split the oversight of the owner address in the contract. Since the entity which controls the privte key of the smart contract is able to perform update operations which can lead to history forgery. Therefore the key usermanagement of the smart contract can be used as a next steps in further research. For this thesis i assume that the private key is well managed andeach transaction coming from the master address is legitim.

5.5 Transaction Throughput

Important because archive can have bulks over 1 million objects

5.6 Deployment

I decided to use truffle to develop, test and deploy the smart contract for the fixity-storage. Truffle is a development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine. Truffle 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 steps I even used the user-interface "Ganache" to get a better feeling for the ethereum blockchain. It allows you to click through your smart contract and look at the state 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 no suited to do 10.000 transactions in a python for loop. Therefore I only used 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 resistent and showed no weakness when persisting 10.000 objects. Truffle also allows to config other networks, e.g. the ropsten test network, which can be defined as a parameter in the deployment process. The deployment requires to have some ether token in your account to pay the miner to integrate your smart contract in a block.

CHAPTER 6

Pooled Testing

6.1 Pooled Testing

Pooled Testing was first introduced by [**?**] as a strategy to screen a large number of military recruits for syphiclis 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. Individuals from negative pools would be declared negative, and specimens from positive pools would be retested individually to identify which recruits had contracted the disease. Dorfmanss motivation for using group testing was to reduce testing costs while still identifying all syphilitic-positive recruits. Today, this would be described as the "case identification problem," because the goal is to identify all positive individuals among all individuals tested. Dorfmanss approach to case identification can be viewed as a two-stage hierarchical algorithm. In this approach, non-overlapping pools are tested in the first stage and individuals from positive pools are tested in the second stage. When the corruption prevalence is small, higher-stage algorithms have proven to be useful at further reducing the number of tests needed [**?**, 1].

6.2 Hierarchical pooling algorithms

6.3 Non-Hierarchical pooling algorithms

6.4 Two Stage Hierarchical Pooling Algorithm

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 needied, 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 [**?**, 3]. The optimal pool size is the size that minimizes the total number of tests needed.

6.5 Parameter Definition

Let p prevalence of corruption, the probability that an object will be corrupted during the preservation process
1-p the probability that the integrity of an object is preserved during the preservation process
$(1-p)^n$ the probability of obtaining a uncorrupted result from a pool of n objects
$1-(1-p)^n$ the probability of obtaining a corrupted result from a pool of n objects
N/j the number of pools of size j in a population of size N
p'N/j the expected number of corupted pools of n in a poluation of N with a corruption rate of p
$E(T) = N/j + n(N/n)p'$ the expected number of tests
[**?**, 3]

6.6 Test Sensivity and Specifity

6.7 Optimal poolsize

There are a few methods to determine the optimal poolsize. The most that I have found during literature review did not consider cost in the retesting stage, meaning that the poolsize could potentionally be N, therefore you get 2 total transactions to do the process, one for ingest and one for the repairing. But then you would have to update your whole archive each time an object is found corrupted, since the pool securing the object is the whole archive. In my experiment, I have to find a way to add a cost to the optimal poolsize to prevent too large pool sizes. The optimal pool size to minimize write transactions on the blockchain is N, because if we can secure every object in the archive with only one write transaction. That would be very inpracticable in a real environment, because you would have to compute a new root hash on each update for each pool. Therefore shooting up the actions with every update for N actions.

CHAPTER 7

Experiment

7.1 Setup

The experiment written in Python in form of a Jupyter Notebook. At first the function for providing the optimal poolsize was implemented, the exact implementation can range froma bernoulli experiment to another form. When the optimal pool sie is found, which is the econd research question I create 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, durng 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 eact method of monitoring the operation cost is by adding up the gas cost, since gas can be transformed nto 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 successfulll I willl deploy the smart contract on the ropsten testnet and add time measurement to the porcess 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 thedummy 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 ffunction also affect the operation cost, since when a corrupted file is found the whole pool has to be re-computed resulting in poolsize + 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 af a table and the experiment can be redone with different poolsies 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.

7.2 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 oject and its respective pool. Its easy to keep the links in the local jupyter notebook but in a industrial environment keeiping those links is rather hard. For now i keep the transaction hash with the pool object

7.3 Object

An object in this experiment is simply a sha256 hash, 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 $\text{pool}_i dis stored in the object's metadata. For the sake of the experiment, t$

7.4 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.

7.5 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) $\text{bulk}_i ngest(2) retrieve(3) get_o bjects_b y_p ool_i d(4) repair.(1$

7.6 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 do scrub each object in the pool with a fresh copy.

7.7 Findings

in the cleaning process i cant count the reall number of corrupt objects because the number gets diluted by the fact that the local pools are false and even when we get a real object we could not recalculate the pool because the pool itself is corrupted therefore an uncorrupted object is seen here as corrupted since the pool cannot prove it anymore a positive sideffect is if you find a corrupt elemen you have to scrub the whole pool, which means that the whole pool has is replaced with fresh copies There are massivele more repairing transactions

CHAPTER 8

Evaluation

The fixity storage ill be evaluated on a local installation of the ethereum blockchain and the online ropsten testnet, where transactions are free of charge. The two key parameters for evaluation are the operation cost and operation time. Operation cost can be computed using the amount of gas used in the whole process and compute the ETH value out of it. The cond method for gatherng the operation cost is by simply checking the amount balance of the master address before and after the experiment. I intent to presenet the operation cost in the form of ethereum and gas because the cost is so dependent on the current price of the Ethereum Currency and other factors, see

8.1 cost

. The second parameter is the amount of time needed to ingest the objects and later retrieve them, in the evaluation the ingestion and retrieval will be handled as two seperate events. The time measuremnt is intereseting because it is expected to be much slower to use the blockchain instead of a third party provider or a local installation of a fixity storage. The resulting time tables will be avarge and total time used for a vertain function, e.g. the avg time used in the client function ingest(). I could also use this to show the cost value of each fjnction and the average cost to use a function of the storage.

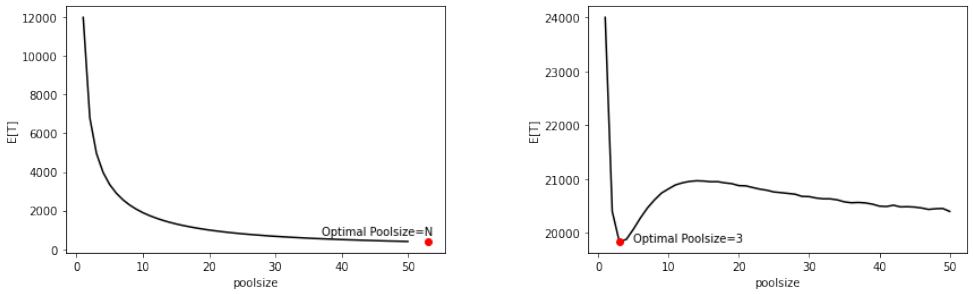
8.2 RQ 1 To what extend can pooled object hashes increase the transaction throughput and reduce cost for a fixity information storage service on the Ethereum blockchain?

8.3 RQ 2 What is the optimal pool size based on the corruption rates of digital objects in the archive in terms of transaction throughput and cost?

I used two metrics to define the cost and transaction throughput of the fixity storage. The first metric is the expected cost, which is the amount of write transactions on the ethereum blockchain. The second metric is the expected throughput, which translates into the total amount of actions needed from ingest to retrieval. The expected throughput acts as a counterweight for the expected cost, because the upper bound of pool size is N when you only account the writes as causation for cost, which can be seen in Figure **??**. Which makes sense, because you would only have two writing transaction on the blockchain if the pool size is N, the first on ingest and the second when you want to update a single object in the archive where you have to recompute a hash over N objects and persist the resulting pool hash on the blockchain. In eq. **??** can be seen that with an poolsize of N, the expected amount of writes is 2. The first term is the number of pools which is calculated by $ceil(N/k)$ and the second term is the amount of positive tested pools which is calculated by multiplying the probability of a pool of size k being negative at a prevalence of p with the amount of pools.

$$E[C] = J + J_{+} \tag{8.1}$$

$$E[C] = \lceil N/k \rceil + ((1 - (1 - p)^k) * \lceil N/k \rceil) \tag{8.2}$$



(a) Expected Cost (b) Expected Throughput

Figure 8.1: Comparison of optimal poolsizes

In Figure **??**, the effect of adding the re-tests and hashing actions can be seen. There is a global minimum of the function where I have the least amount of actions needed for the whole preservation process. This effect can be explained through picturing a retrieval of a corrupt pool where I have to re-test and rehash the number of objects in the corrupt pool, and the bigger the pool is, the more actions I have to make in order to restore the integrity of the archive. I have found the optimal poolsize regarding the throughput by minimizing eq. **??**.

$$E[T] = N + J + J_+ + J_+ * k \tag{8.3}$$

Eq. **??** where N number of times I have to hash the initial ingest bulk, J is the number of pools, J is the number of positive pools and $J_p * k$ is the number of times I have to recalculate a hash for an object in a positive pool I count the number of initial hashing actions N, I also count the number of writing-actions which is exactly the number of pools existing,third I count the number of positive pools which have to be re-written onto the blockchain and as last I cound the number of objects which have to be re-tested in order to find the corrupted object in the pool. And because I counted in the number of hashing actions, the function to define the optimal pool size does go N anymore. So the optimal pool size regarding the cost is N, and the optimal poolsize regarding the transaction throughput is the local minima of equation **??**.

CHAPTER 9

Discussion

CHAPTER 10

Conclusion

CHAPTER 11

Related Work

Enter your text here.