26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Endolith: A Blockchain-based Framework to Enhance Data Retention in Cloud Storages

Thomas Renner, Johannes Müller, Odej Kao
Technische Universität Berlin, Germany
{firstname.lastname}@tu-berlin.de

***Abstract*—Blockchains like Bitcoin and Ethereum have seen significant adoption in the past few years and show promise to design applications without any centralized reliance on third parties. In this paper, we present *Endolith*, an auditing framework for verifying file integrity and tracking file history without third party reliance using a smart contract-based blockchain. Annotated files are continuously monitored and metadata about changes including file hashes are stored tamper-proof on the blockchain. Based on this, Endolith can prove that a file stored a long time ago has not been changed without authorization or, if it did, track when it has changed, by whom. Endolith implementation is based on Ethereum and Hadoop Distributed File System (HDFS). Our evaluation on a public blockchain network shows that Endolith is efficient for files that are infrequently modified but often accessed, which are common characteristics of data archives.**

I. Introduction

Data volume is growing at an increasing rate. Besides the large amount of new data, a lot of already existing data needs to be stored safely and reliably for a long period of time. Reasons for a growing demand for long-term storage solutions are the rise of digitalization and a trend towards more organization policies, government laws and regulations for retaining data for specified periods of time. For instance, some companies are required to keep tax information, contracts, and business reports for up to ten years depending on the country they operate in [1]. Another example concerns research data and results that should be publicly available for a long period of time. These policies result not only in more data volume that needs to be archived. Moreover, ensuring data integrity, validity, and provenance becomes an important, but also a challenging task [2]. For instance, proving that a file stored long time ago has not been changed without authorization or, if it did to track when a file has changed by whom.

Most organizations' data is stored on dedicated storage infrastructure consisting of distributed file systems, storage area networks, or cloud object storages. Additionally, automated storage tiering [3] [4] is often used to increase storage capacity and performance, but also to meet long-term requirements due to government or industry regulations. Thereby, data that is infrequently accessed is automatically moved to a separate storage tier or system designed for long-term archival purposes. Some archival storage systems such as Amazon Glacier [5] follow a Write Once, Read Many (WORM) paradigm, which treats all archive data as read-only to protect it from modification. While other data archiving systems, like Google Coldline [6] and HDFS Archival Storage [7], are more flexible and allow modifications. Even if these systems provide high fault tolerance, in a worst case scenario, data, as well as auditing data that summarizes operations on that data, can be accidentally or maliciously modified or deleted without notification [8].

Blockchain technologies and their respective peer-to-peer networks have attracted a lot of attention recently. Prominent examples include Bitcoin [9] and Ethereum [10]. A blockchain, also known as distributed ledger, is a tamper-proof append-only list of data transactions that are linked and secured using cryptography. Additionally, it consists of a peer-to-peer network that collectively keeps the ledger up to date, and collectively validates transactions and generates new blocks storing those transactions. Blockchain technology provides high trust that transaction are not altered, because any participant can inspect the ledger, but no one controls it. Thus, a blockchain can enable trust between participants without the need of a central third party. Additionally, some blockchains like Ethereum allow to execute programs on the blockchain, also known as smart contracts [11], without any possibility of downtime, censorship, fraud or third-party interference.

This paper presents Endolith, an auditing framework to support file integrity, longevity, and history tracking in storage systems. Endolith integrates blockchain technology into storage infrastructures to enable these features trustfully without the need of a central authority. It allows to monitor annotated files and stores necessary metadata about file changes tamper-proof on a blockchain. In particular, once integrated into an existing storage system, it continuously hashes the selected files on creation and modification. The resulting hash, modification time and user will then be persistently stored on the blockchain. Endolith is based on a smart contract-enabling blockchain. The deployed smart contracts use the metadata for their validation and history tracking functionality. Additionally, Endolith's smart contracts provide functionality to indisputably proof whether a file existed at a given time or not. It is important to mention that Endolith does not store any file content on the blockchain.

Endolith's implementation and evaluation are based on Ethereum, a smart contract-based blockchain and Hadoop Distributed File System (HDFS) [12]. However, due to decoupling via an introduced Storage API, it is possible to integrate additional storage systems and automated tiering solutions to Endolith. We evaluated the overhead of Endolith

0-7695-6380-5/18/$31.00 ©2018 IEEE
DOI 10.1109/PDP2018.2018.00105

on Ethereum's official testnet and a 10 node HDFS cluster. The overhead of creating and modifying files in terms of additional response time for the user is 42.5 seconds. Whereas most of the time was consumed for writing and confirming a transaction on the blockchain. The overhead of validating files is between 0.54 to 1.20 seconds, because the functionalities can be performed locally without writing any transaction. Due to the high overhead of file creation and modification with Endolith, we introduce a transaction manager that allows to continue using HDFS, even when transactions are not yet written and confirmed by the blockchain network. Nevertheless, Endolith focuses on data, which is less frequently modified, because storing data on a blockchain needs to be done economically due to its high costs.

Contributions. The contributions of this paper are:

- An approach that supports file integrity, longevity, retention, and history tracking.
- An implementation of our approach, which we call Endolithm, that is integrated with Ethereum and HDFS.
- An evaluation on the Ethereum testnet blockchain showing the overhead of Endolith.

Outline. The remainder of the paper is structured as follows. Section II presents the background. Section III presents the system design. Section IV presents the work flow of using Endolithm. Section V presents our implementation. Section VI presents our evaluation. Section VII presents related work. Section VIII concludes this paper.

II. Background

This section describes the background of blockchain technology and smart contract-based blockchains.

A. Blockchain Technology

A blockchain, also known as shared ledger, is an append-only list of records, which are linked and secured using cryptography. Prominent blockchain implementations include Bitcoin [9] and Ethereum [10]. Writes to the blockchain are called transactions, whereby multiple transactions are grouped and permanently stored into a single block. These blocks are chronological and linear connected to form a blockchain, similar to a linked-list. Each block within the blockchain is uniquely identified by a hash of the block header. A block header includes the Merkle Tree [13] root of all transaction hashes stored in the block. Additionally, each block stores the hash of a previous block header as a reference. Thus, the sequence of hashes linking to its previous block creates a chain going back all the way to the first block ever created. Figure 1 illustrates the structure of a blockchain.

A blockchain is managed by a private or public peer-to-peer network that collectively validates and generates new blocks. Therefore, the blockchain network consists of multiple nodes, where each node has a local copy of the blockchain. Some nodes participate in a leader election process (i.e. proof-of-work) that determines which node gets the privilege to append the next block to the chain. These nodes, which are actively competing to become the leader of the next round, are called

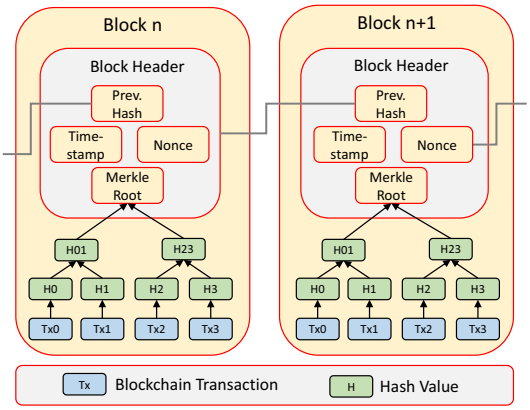

Fig. 1: Data structure of a simple blockchain.

miners. At the start of each leader election round, all miners start working on a new computational problem (e.g. producing a hash) that depends on three pieces of data: new blocks of transactions, the last block on the blockchain, and a random number. This refers collectively as block header for the current block. Each time a miner performs the hash function on the block header with a new random number, they get a new result. To win the election process, a miner must find a hash that begins with a certain number of zeros. Just how many numbers of zeros are requires is a shifting parameter determined by how many miners and how much computing power is attached to the network. The first miner that solves the problem gets the privilege to write the new block with pending transactions that are not yet included in any block. Motivation for participating and winning the election is a monetary reward. The winners may issue themselves an amount of the mined currency and they get to collect all fees that are charged for transactions. In order to have their transaction prioritized, users may offer to pay a higher fee. As a result, the blockchain forms a system that enables decentralized consensus without the need of any central authority.

B. Smart Contract-Based Blockchain

Smart contracts are computer programs made to verify, or enforce the negotiation or performance of a contract. They are automatically enforced by the consensus mechanism of the blockchain, without relying on a trusted authority. The consensus protocol of the blockchain has the goal to ensure the correct execution of smart contracts. Ethereum is a prominent smart contract based blockchain. Its underlying Ethereum Virtual Machine (EVM) has a notion of global state including accounts, balances and storage. Each transaction must ensure a valid transition from the canonical state preceding a transaction to the new state it leaves the EVM in. Therefore the order in which transactions are processed and written into a mined block is crucial. All EVM transitions are executed by every participant of the network and stores new states on the blockchain. The smart contracts inherit the blockchain properties of decentralization, zero downtime and security against

censorship or fraud. Smart contracts are written in Solidity, a Turing-complete bytecode language, which compiles into a special assembly code that can be interpreted by the EVM. Therefore, a contract is a set of functions, each one defined by a sequence of bytecode instructions.

III. Design of Endolith

Endolith is a framework to support data retention and archiving as well as file verification and history tracking in existing storage infrastructures. Therefore, it monitors files that are annotated by a user or programmable rules and policies that dictate if and when data can be annotated - similar to policies known in automated storage tiering that move data from a performance to a capacity storage. Afterwards, selected metadata of the annotated files are written to the underlying blockchain via smart contracts. Endolith automatically generates and deploys these smart contracts on the blockchain providing auditing functionality and secure access to the metadata stored on the blockchain. The functionality includes proof of existence, file validation, and history tracking. It is important to emphasize that Endolith suits best for files that are only infrequently modified. This is because it takes a certain time until a transaction including the metadata or smart contract creation is mined and confirmed on the blockchain.

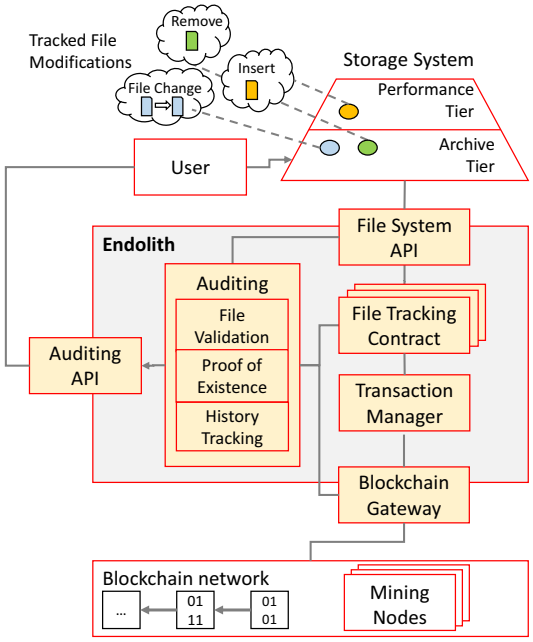

Fig. 2: Overview of Endolith and its components.

Figure 2 gives an overview of Endolith and shows how it is integrated with existing systems. The remainder of this section describes Endolith's components in more detail.

Storage User. The user stores its files on the storage infrastructure. Files can be annotated by the user or storage provider to be tracked with Endolith. For instance, when it must be stored reliably for a long time period. Operations such as file creation, changes, moves, and deletes on these annotated data are monitored and sent to Endolith including additional file metadata such as file hash, user name, and modification time. All this data are then stored via Endolith specific smart contracts on the blockchain, however only readable and accessible for users with explicit rights. Similar to public-key encryption, only users whose public key is defined in the file tracking smart contract can access the data with their private key. Users with no access to the smart contract can only see data stored in the smart contract and its code in form of raw binary data within transactions on the blockchain network.

Storage Provider. The provider runs the storage infrastructure to offer storage to its users. The provider can offer a trustful long-term storage service by integrating Endolith. This is because the current state and previous states of a file and its metadata about who changed the file and when changed it are stored on the underlying blockchain. Based on the tamper-proof and traceable characteristics of the blockchain, the provider can guarantee to its users that a file stored long time ago has not been changed unauthorized, or to track who has and when has a file changed. Additionally, it is conceivable that the provider can use the data for detecting intrusion or anomalous behavior.

File System API. The File System API provides methods for adding information about file creation, changes, and deletion to Endolith. Additionally, it can be used to add information from Endolith to existing files. For instance, to add the smart contract address as metadata to its corresponding file, which allows an easier association between both. The File System API can also be used by a storage provider to automatically monitor their storage systems by using hooks that listen to file operations and then automatically use the API to forward the collected data to Endolith. Similar to an automated tiered storage, where cold data is moved to a separate long-term storage tier based on access times, files could be automatically annotated and tracked with Endolith when moved to a long-term tier.

Blockchain Gateway. The gateway provides access to a blockchain network, which can either be a private or public. The gateway participates in the blockchain network and thus, has a local copy of the blockchain. However it is decoupled from the mining network and thus does not act as a miner to perform the proof-of-work algorithm. The reason for that is to save compute resources for Endolith's main tasks. First, sending contract creation and execution transactions to the blockchain network, where they are mined and executed. Second, some smart contract functions are directly executed on the gateway, as they do not modify the global state of the blockchain. These functions operate on the local copy of the blockchain and return fast results

Transaction Manager. The transaction manager automatically generates and deploys smart contracts on the blockchain network. Therefore, it deploys one specific smart contract for every selected file that should be tracked. Additionally, it converts data provided by the File System API, such as file

changes, into a smart contract executable format. In the other direction, it translates transaction receipts to a File System API capable file format. Additionally, it buffers transaction if needed. This is because it takes time until a transaction becomes available on the blockchain. Instead of waiting until the transaction is available and verified, we collect the status and periodically check if the transaction is successfully written to the blockchain. If a transaction is rejected by the blockchain, the transaction is submitted again. Following transactions that effect the same file are queued in a FIFO queue and submitted after the ancestor was successfully written to the blockchain.

File Tracking Contract. We designed a Track Change Contract template, whereof Endolith automatically deploys an instance of for every new annotated file. A smart contract instance is responsible for tracking all changes of exactly one file. Section V describes the contract setup including its attributes and functions in more detail. Due to scalability requirement of archiving a large amount of files, it is not feasible to store all meta data in one global smart contract containing all data. This is because it would take a lot of time to index that contract and there is a limitation in terms of size for a contract. After a new file is archived and its related smart contract is deployed, its smart contract address is stored to the file meta data to find the corresponding file on the block chain. One could also store this in a dedicated smart contract, file or database.

Auditing. Auditing provides functionalities for File Validation, Proof of Existence, and File History Tracking. These functions are embedded in smart contracts and are executed locally on the node running Endolith. All auditing functions are executed on a up-to-date copy of the blockchain and do not change the state of the contract, and thus can be executed without the need of writing a transaction on the blockchain.

- **File Validation** ensures that the file is correct. The user can validate if the local and remote files are the same based on the local and remote baseline hash stored on the blockchain. This can be useful after uploading or downloading a file from the storage system. Therefore, all change entries stored on the blockchain include timestamps.
- **Proof of Existence** verifies the existence of a file at a specific time without the need of a central authority. Endolith allows to validate files by comparing the local and blockchain hash. Because Endolith doesn't store the file content on the blockchain, it is anonymous.
- **File History Tracking** refers to the process of tracing and recording files with explanation of how it got to the present state. Therefore, every modifications on selected files are immutably recorded on the blockchain network. Endolith uses the information to provide track change history of a file record.

Auditing API Auditing functions are accessible for a user via an API. The user access is based on public key authentication. The public keys are stored in the generated contracts.

IV. Workflow of Endolith

This section describes the workflows of Endolith for file tracking, validation, and history tracking in more detail.

A. File Tracking With Endolith

This section describes the workflow of monitoring files with Endolith. The workflow is shown in Figure 3 and consists of following steps:

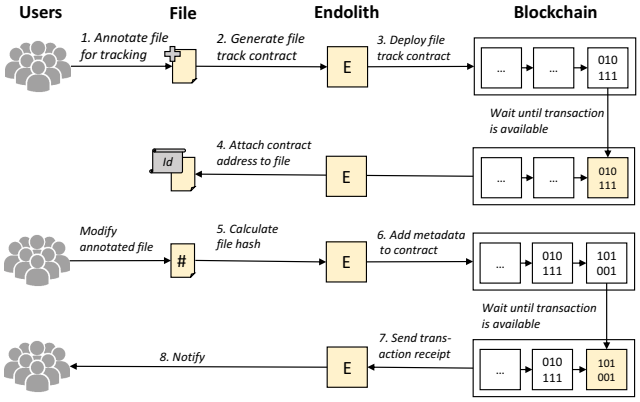

Fig. 3: Workflow of annotating and monitoring with Endolith.

1) The user annotates a file for long-term file tracking.
2) A dedicated smart contract for that file is automatically generated based on Endolith's file tracking template.
3) The generated file tracking contract is submitted to the blockchain.
4) A unique smart contract address is returned after it is successfully deployed on the blockchain. This contract address is attached as meta data to the file residing in the storage system to make a connection between the smart contract and the file.
5) The file hash is calculated using a cryptographic one-way hash function. Additionally, user name, and modification time is collected from the storage system. It is important to emphasize that hashing is not done within the smart contract on the blockchain. This would be expensive and even impossible for large files, due to the file data that needs to be send to and processed on the blockchain network.
6) All data is parsed to the corresponding file contract, which is identifiable by the contract address attached as metadata to the file.
7) Endolith receives a receipt confirming that the transaction was successfully written to the blockchain.
8) Afterwards, once the transaction is successfully written to the blockchain, the user is notified and, if additional queued transactions for that annotated file exists, they are submitted to the blockchain.

Step one to four is only done once, when the file is annotated. Step five to eight are repeated every time a file changes. In order to reduce the waiting time until a transaction is available on the blockchain, subsequent transactions are

queued in the transaction manager, which is described in Section V-B in more detail.

B. File Validation With Endolith

This section describes the workflow of validating a tracked file with Endolith. The workflow is shown in Figure 4.

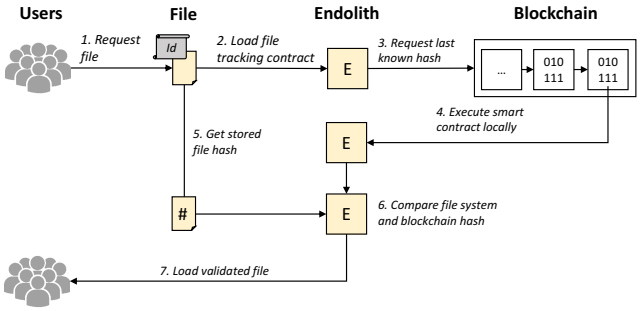

Fig. 4: Workflow of validating a file with Endolith.

1) The user requests a file from the file system.
2) The corresponding file tracking contract is loaded based on the as metadata attached smart contract address.
3) Endolith requests the last known hash from the loaded file tracking contract.
4) The smart contract validate function is executed locally by Endolith on the up-to-date copy of the blockchain, to get the last known hash stored on the blockchain for that requested file.
5) The hash of the file version stored in the storage system is calculated.
6) The current resulting hash value and the last value stored in the smart contract are compared to each other. If both values match, the file is presumed to be the same, because the probability of an accidental hash collision, which means that a file will hash to the exactly value, is very small.
7) The user loads the validated file.

Additionally, Endolith allows its users to validate their local copies of a file against the ground truth stored on the blockchain network without touching the remote file. Also, he or she can validate, if the local version existed at any time. Another advantage of this hash-based file verification is its good performance without the need of comparing bit per bit and it is possible to compare files without making its content visible to anyone. Also the hash size is relatively small in size compared to the file size, and thus is suitable to be stored on the blockchain.

C. History Tracking With Endolith

By storing the file hashes including additional file attributes maintained by the storage system, such as owner, time stamp, and URI, on a blockchain using dedicated smart contract, it is possible to guarantee and exactly determine when and how a file has being changed at any moment in time. If an already stored file is modified, the updated file attributes including the new file hash is appended to the smart contract and is stored the blockchain network. As soon as the transaction is mined and confirmed, we can indisputably prove that the file record in the storage exists and securely track who and when changed a file, because transactions on the blockchain cannot be modified or deleted. Additionally, all that functionality can be done without any third party interference.

V. Implementation

This section describes Endolith implementation using Ethereum as smart contract-based blockchain and HDFS as a storage system. First, the implementation of the file tracking smart contract is explained in more detail. Afterwards, the implementation of the transaction buffer is discussed.

A. File Tracking Contract

When a file is annotated for long-term tracking and validation, a corresponding smart contract is automatically generated and deployed to the Ethereum blockchain network. A file tracking contract includes the current and all past file hashes of the tracked file including its modification time and user. It is automatically derived from a generic template, which is shown in Figure 5. A file tracking contract belongs to exactly one file, and thus, for every annotated file exactly one file tracking contract exists on the blockchain. It is important to emphasize that a file Uniform Resource Identifier (URI) of a file tracking contract can change due to file renaming or moving. Therefore, after a file tracking contract is successfully deployed on the blockchain, the unique smart contract address is attached to the corresponding file in HDFS as a metadata attribute. Afterwards, every file modification including its new file hash, modifying user, and time is stored on the blockchain using its corresponding smart contract, which is identifiable by the smart contract address attached as file metadata. We used the distributed file checksum algorithm that HDFS has built in, which returns a single fixed length MD5 checksum quickly for any size of file independent from the number of blocks.

An instance of a file tracking contract consists of following attributes and functions:

- **fileURI**. The file's current URI, which can be changed when a file name, path, or the storage system changes.
- **modifications**. A list of the current and all past file hashes of the tracked file including its creation time, user, and fileURI.
- **currentHash** is a cache variable that stores the last known hash to speed-up requests for validation.
- **updateFile(FileURI, FileHash)** Writing an update to the smart contract requires the files path, a new path may be set if the file moved, and the new file hash. All necessary fields are updated and new values are pushed onto the list of modifications. This operation changes the global state of the contract and results in a transaction.
- **validate(FileHash)** The validate function verifies that any given hash equals the current value stored on the blockchain and returns a boolean value accordingly. The validate call does not alter the state of the EVM and

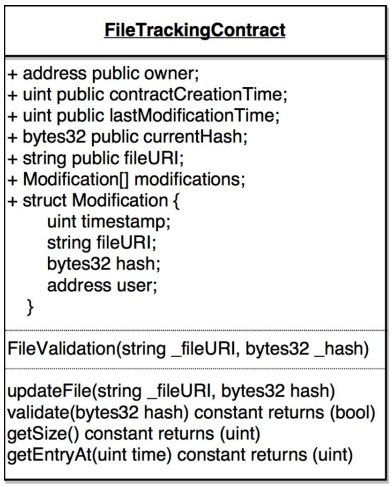

Fig. 5: Overview of File Tracking Contract Attributes and Functions.

therefore does not require a transaction to be sent and can be performed locally.

- **getEntryAtTime(Time)** Any hash for any point in time since contract creation can be retrieved from the contract and used to analyze modification history

B. Transaction Buffer

The time until a submitted transaction for contract creation or execution is mined and confirmed by the blockchain network can take up several seconds. Therefore, Endolith is not suitable for so called hot data, which is frequently modified. However, we implemented a Transaction Buffer that queues waiting transactions. Additionally, we collect the status of pending transactions and periodically check if the transaction is successfully written to the blockchain or not. If not, which is sometimes the case especially on public blockchain networks, the transaction is resubmitted.

We estimate the confirmation time ct of a transaction tx by $\Delta ct_{(tx)} = (\Delta bt * c) + \Delta \frac{bt}{2}$. Let Δbt is the average block time that the blockchain network takes to generate a new block. c is the number of subsequent blocks that must confirm that the transaction is valid and is part of the blockchain. $\Delta \frac{bt}{2}$ the estimated pickup time between transaction submission and transaction being mined in a block, assuming the transaction gets mined in the next mined block. For instance, on the Ethereum main net the average block time bt is 15 seconds[1] The Ethereum whitepaper suggest to wait 7 block confirmations [14]. As a result, we estimate the response time of a transaction with 82.5 seconds.

Endolith holds a list of all pending contracts and periodically checks if the creation is successfully deployed. In order to allow modification on that file during its waiting time, all subsequent changes are stored in a dedicated waiting queue for that file. Once the contract creation transaction is confirmed, the contract address is received and stored as metadata to the file and is used to write all updates to the corresponding tracking contract. We implemented this as a FIFO queue, when a transaction is successfully written and confirmed, the next transaction is submitted. In blockchain networks the order of transactions is important. For instance, to avoid replay attacks, in which a valid transaction is maliciously over and over to the network. Therefore, we buffer transactions in memory and wait until the predecessor was confirmed.

[1]https://etherscan.io/chart/blocktime, accessed 2017-11-01

VI. Evaluation

This section presents our benchmark setup and results.

A. Experimental Setup

All experiments were done using a 11 node cluster. Each node is equipped with a quad-core Intel Xeon CPU E3-1230 V2 3.30GHz, 16 GB RAM, and three 1 TB disks with 7200RPM organized in a RAID-0. All nodes are connected through a single switch with a one Gigabit Ethernet connection. Each node runs Linux, kernel version 3.10.0, and Java 1.8.0. 10 Nodes run Hadoop HDFS 2.7.1 with default configuration. One Node runs Endolith and Geth[2] version 1.7.1 as gateway to Ethereum's public testnet Ropsten[3].

B. Benchmark Result

This section describes the benchmark results. First, the response time of writing and reading a file with Endolith is measured. Afterwards, the additional costs of using Endolith are reported. We repeat all experiments seven times and report the median.

Writing Response Time. This benchmark measures the response time of writing with and without Endolith. Therefore, we generate random files with a fixed file sizes ranging from 64 MB to 8192 MB by using dd, a command-line utility for Unix and Unix-like operating systems. Figure 6 reports the results of writing files with varying sizes. The blue bar represents the response time until the file is available in HDFS. The orange bar represents the response time until the metadata is collected and the transaction is send and confirmed by the Ethereum network. The median response time for collecting the metadata as well as sending and confirming the contract transaction is 42.5 seconds. This response time is approximately the same independent of the file size, because the opcode and input data for deploying a file tracking smart contract has approximately the same size independent of the file size. However, maximum and minimum values are far from each with 15.590 seconds and 105.843 seconds. This is because block time varies heavily on the testnet. Collecting the metadata and sending the contract creation took 0.7 seconds on the average. Transaction confirmation takes most of the response time. The average block time on the Ropsten testnet is between 10 and 15 seconds, as block confirmation value we used 5, which is the default value in Geth. When the file size increases, the proportion of the overhead caused by Endolith

[2]https://github.com/ethereum/go-ethereum, accessed 2017-11-01
[3]https://github.com/ethereum/ropsten, accessed 2017-11-01

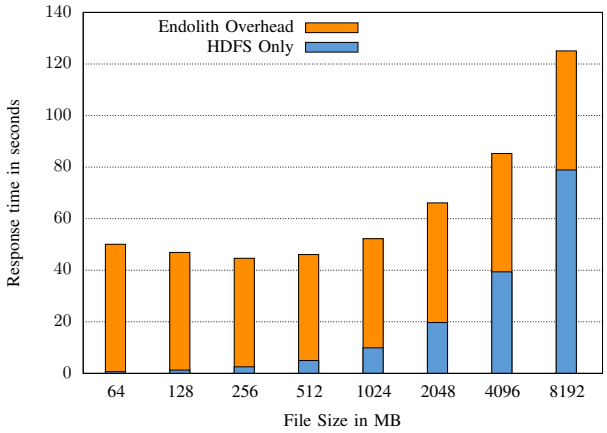

Fig. 6: Response time of writing a file with varying size to HDFS with and without Endolith.

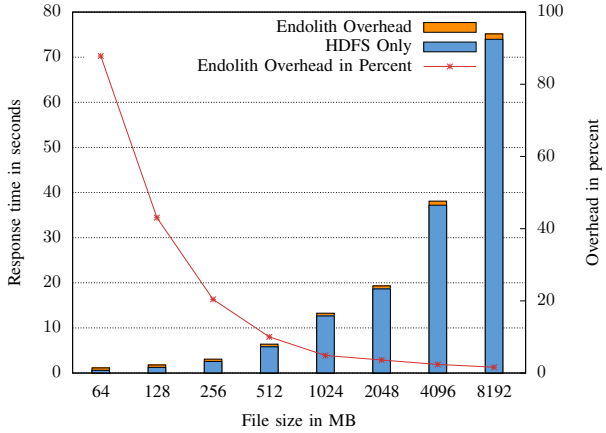

Fig. 7: Response time of reading and validating a file with varying size from HDFS with and without Endolith.

decreases, because more time is spent on the file transfer. It is important to emphasize that Endolith allows the user to use HDFS during that time due to the Transaction Buffer described in Section V-B. However, due to the long response time until a transaction is confirmed, Endolith is more suitable for long-term data, which is modified rarely or not at all.

Reading and Validating Response Time. Similar to the writing benchmark, we generate random files with varying sizes between 64 MB and 8192 MB. Figure 7 reports the results of reading and validating the files with varying sizes. Before reading the file with Endolith, we modify it 50 times to generate data within the corresponding file tracking contract. The blue bar represents the response time until the file is copied from HDFS to the local disk. The orange bar represents the response time until the local file is hashed and validated against the value stored on the ethereum blockchain. The overhead varies between 0.54 to 1.20 seconds. The red line represents the overhead in percent, which varies from 87.83% to 1.62%. When the file size increases, the overhead caused by Endolith decreases, because more time is spent on the file transfer. In comparison to the write response time, the reading and validating overhead is small. This is because the smart contract execution for retrieving the file hash can be done locally without writing any transaction on the blockchain network.

Costs of Creating And Modifying a File This benchmark measures the cost of creating and modifying a single file in terms of gas [14]. Gas is the name for a special unit used in Ethereum. It measures how much work an action or set of actions takes to perform. Every operation that can be performed by a transaction or contract on the Ethereum platform costs a certain number of gas, with operations that require more computational resources costing more gas than operations that require few computational resources. We generated a 1024 MB file, loaded the file to HDFS, and repeated both steps 10 times for overriding the file in HDFS. When a file is created, a corresponding smart contract is deployed. This first step costs 147060 gas. Afterwards, every modification costs constantly 132060 gas. This is because exactly the same number of operations are executed on the ethereum network. It is important to emphasize that the gas consumption on the ethereum test net can differ from the main net. Additionally, users should be thoughtful which and how many files to track when using Endolith on a public network, because executing code on the blockchain generates costs.

VII. Related Work

This section presents related work for using blockchain technology in storage infrastructures.

Ghoshal and Paul [15] present an auditing scheme for cloud data that requires no third party involvement. Similar to Endolith, their approach allows to ensure and check integrity of selected file based on blockchain technology. In particular, they propose to split the selected files into fixed-length file-blocks, hash the file-blocks, and generate a Merkle Tree per file based on all file-blocks hashes. The Merkl Tree root, the previous block hash, and other file metadata are then appended as a new block to the blockchain. In order to speed up the verification process, they use on a leaf number-based verification technique. The authors report that their approach suits only well for files with rare updates, because changing a file leads to a modification of the corresponding block, and thus, any subsequent block needs to be rewritten, which is time and compute intensive. In comparison, we append the file metadata as a transaction to the blockchain and do not change any block due to transparency reasons. Additionally, their approach is not suitable for a public blockchain network, where modifications on already written and confirmed blocks are explicit unintended.

ProvChain [16] is a data provenance system for cloud storage systems using blockchain technology. It monitors all operations on files and publish them to a blockchain.

ProvChain hashes data operations, constitute a Merkl tree, and anchors the root node into a blockchain transaction. By this, it is possible to guarantee that data provenance was not temper. ProvChain tracks files on operation level including read, move, and copy operations, and do not offer a functionality for a content-based file validation.

Blockstack [17] presents a blockchain-based naming and storage system. They present their knowledge about running a public key infrastructure service on top of Namecoin and how they migrate to bitcoin. In Blockstack, users can register and securely associate data with them. Only the owner of the particular private key can write or update the name-value pair.

Other systems like Filecoin [18], Permacoin [19], Storj [20], and SIA [21] aim to replace cloud storage. Instead of rewarding miners for offering compute resources, they reward miners for offering storage (i.e. proof-of-storage).

ETH Drive [22] runs IPFS [23], an peer-to-peer distrbuted file system, and uses Ethereum as blockchain to provide tamper-proof data provenance to check data integrity.

VIII. Conclusion & Future Work

This paper presents Endolith. Endolith integrates blockchain technology into storage infrastructures. It tracks annotated files and stores selected metadata of these files on the blockchain by using smart contracts. Additionally, Endolith's smart contracts provide auditing functionalities for tamper-proof file validation and history tracking. This way, Endolith supports long-term storage and ensuring data integrity. Endolith focuses on archival data and allows to prove that a file stored long time ago has not been changed without authorization or to track when a file has changed by whom. Since it takes time until a transaction is written and confirmed on a blockchain, Endolith is not a good choice for files that are modified high frequently. Endolith is best applied for archival or document storages, where files are infrequently modified and user response time for creating or modifying a file is not critical.

In the future, we want to improve user access management and introduce file encryption on the blockchain for a secure document transfer. Additionally, we plan to integrate other systems beside HDFS to show that Endolith can track files even beyond file system boundaries. Also, we want to evaluate Endolith more systematically by using additional evaluation criteria. Especially, we want to find out how Endolith performs on a private blockchain with a lower block time. And, how it performs having many file changes at the same time. Additional, using a private blockchain instead of a public blockchain allows being independent from price fluctuations and speculations. Nevertheless, our evaluation on the Ethereum testnet and HDFS shows that Endolith is a effective framework to improve data retaining in existing storage systems with an acceptable overhead.

Acknowledgments

This work has been supported through grants by the German Science Foundation (DFG) as FOR 1306 Stratosphere and by the German Ministry for Education and Research (BMBF) as Berlin Big Data Center BBDC (funding mark 01IS14013A).

References

[1] Ernst & Young, "Data Retention and Preservation - Overview on Requirements in Selected Countries," 2014, http://www.ey.com/Publication/vwLUAssets/Data_retention_and_preservation/$FILE/EY-Data-Retention-brochure.pdf (accessed November 2017).

[2] Y. L. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in e-Science," *SIGMOD Rec.*, vol. 34, no. 3, pp. 31–36, Sep. 2005.

[3] Y. Cheng, M. S. Iqbal, A. Gupta, and A. R. Butt, "CAST: Tiering Storage for Data Analytics in the Cloud," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: ACM, 2015, pp. 45–56.

[4] K. Krish, A. Anwar, and A. R. Butt, "hatS: A Heterogeneity-Aware Tiered Storage for Hadoop," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 502–511.

[5] "Amazon Glacier," https://aws.amazon.com/glacier/ (accessed November 2017).

[6] "Google Coldline," https://cloud.google.com/storage/archival/ (accessed November 2017).

[7] Hadoop, "Archival Storage, SSD & Memory," https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/ArchivalStorage.html (accessed November 2017).

[8] B. Lee, A. Awad, and M. Awad, "Towards Secure Provenance in the Cloud: A Survey," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Dec 2015, pp. 577–582.

[9] S. Nakamoto, "Bitcoin: A Peer-to-peer Electronic Cash System," 2008, http://www.bitcoin.org/bitcoin.pdf (accessed November 2017).

[10] G. Wood, "Ethereum: A Secure Decentralised Generalised transaction ledger," 2014, http://gavwood.com/Paper.pdf (accessed November 2017).

[11] N. Szabo, "The Idea of Smart Contracts," *Nick Szabos Papers and Concise Tutorials*, 1997, http://szabo.best.vwh.net/smartcontractsidea.html (accessed November 2017).

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1–10.

[13] R. C. Merkle, "Protocols for Public Key Cryptosystems," in *Security and Privacy, 1980 IEEE Symposium on*. IEEE, 1980, pp. 122–122.

[14] Ethereum White Paper, "A Next-Generation Smart Contract and Decentralized Application Platform," https://github.com/ethereum/wiki/wiki/White-Paper (accessed November 2017).

[15] S. Ghoshal and G. Paul, "Exploiting Block-Chain Data Structure for Auditorless Auditing on Cloud Data," in *Information Systems Security*. Springer, 2016, pp. 359–371.

[16] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A Blockchain-based Data Provenance Architecture in Cloud Environment With Enhanced Privacy and Availability," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 468–477.

[17] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A Global Naming and Storage System Secured by Blockchains," in *USENIX Annual Technical Conference*, 2016, pp. 181–194.

[18] Filecoin, "A Cryptocurrency Operated File Network," *Tech report*, 2014, http://filecoin.io/filecoin.pdf (accessed November 2017).

[19] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing Bitcoin Work for Data Preservation," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 475–490.

[20] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a Peer-to-peer Cloud Storage Network," 2014, hhttps://storj.io/storj.pdf (accessed November 2017).

[21] D. Vorick and L. Champine, "Sia: Simple Decentralized Storage," 2014.

[22] X. L. Yu, X. Xu, and B. Liu, "EthDrive: A Peer-to-Peer Data Storage with Provenance," in *Proceedings of the Forum and Doctoral Consortium Papers Presented at the 29th International Conference on Advanced Information Systems Engineering, CAiSE 2017, Essen, Germany, June 12-16, 2017*, 2017, pp. 25–32.

[23] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf (accessed November 2017).