Clouseau: Blockchain-based Data Integrity for HDFS Clusters

Alyzia Konsta*, Ioannis Mytilinis†, Katerina Doka*, Sotiris Niarchos* and Nectarios Koziris*
*Computing Systems Laboratory, National Technical University of Athens
{akonsta, katerina, sniarchos, nkoziris}@cslab.ece.ntua.gr
†École Polytechnique Fédérale de Lausanne (EPFL)
ioannis.mytilinis@epfl.ch

***Abstract*—As the volume of produced data is exponentially increasing, companies tend to rely on distributed systems to meet the surging demand for storage capacity. With the business workflows becoming more and more complex, such systems often consist of or are accessed by multiple independent, untrusted entities, which need to interact with shared data. In such scenarios, the potential conflicts of interest incentivize malicious parties to act in a dishonest way and tamper the data to their own benefit. The decentralized nature of the systems renders verifiable data integrity a strenuous but necessary task: The various parties should be able to audit changes and detect tampering when it happens.**

In this work, we focus on HDFS, the most common storage substrate for Big Data analytics. HDFS is vulnerable to malicious users and participating nodes and does not provide a trustful lineage mechanism, thus jeopardizing the integrity of stored data and the credibility of extracted insights. As a remedy, we present *Clouseau*, a blockchain-based system that provides verifiable integrity over HDFS, while it does not incur significant overhead at the critical path of read/write operations. During the demonstration, the attendees will have the chance to interact with Clouseau, corrupt data themselves, and witness how Clouseau detects malicious actions.

1. Introduction

Modern business workflows have become increasingly complex and decentralized, allowing different organizations and institutions to share and operate on sheer volumes of data, mostly through distributed storage and processing frameworks. In such environments, the lineage of information usually includes updates by more than one entities, which are not by default trusted, but may in fact act maliciously, tampering data for their own interest. Even within the same company, when various departments access a common datastore, possible data inconsistencies may lead to intra-company disputes that are hard to resolve. For the aforementioned reasons, it is important that distributed datastores provide for verifiable data lineage and integrity.

Until now, distributed data stores mostly assume that participating entities are trusted and always act in an honest way. Most security features introduced to such frameworks revolve around user authentication and authorization, completely neglecting the trust aspect [1], [2]. However, legitimate users and entities can still harm the system.

Ideally, what we need in order to protect shared data is an immutable audit log, that keeps track of all the interactions between the various parties involved as well as a mechanism to concisely prove the validity and integrity of the stored data, without having to retrieve the whole, possibly huge dataset.Thus, any tampering of the data, deliberate or not, will be detected by third-party auditors, and the responsible party will be identified through the logs. To avoid privacy breaches and boost performance, we would not expose the data itself to the third-party auditors, but only a specific set of metadata that could reconstruct state and prove provenance.

Blockchain is a technology that naturally fits the purpose and has recently flourished exactly due to its ability to provide secure transaction tracking. It inherently guarantees immutability and does not rely on a specific trusted third-party but rather on decentralized auditors. Thus, the use of blockchains is nowadays not restricted exclusively to the support of cryptocurrencies but has been adopted by a plethora of applications such as asset management, supply chains and IoT.

Introducing blockchains into databases and storage systems in order to safeguard data is not a new concept. In the last few years, there have been several attempts originating both from academia and the industry towards this direction. Systems like Veritas [3], Spitz [4], ChainifyDB [5] and BigchainDB [6] combine the merits of databases with those of blockchains, offering regular SQL interfaces with rich querying capabilities and immutability at the same time.

However, such approaches opt for radically new designs and require major changes in the software stack of the current data processing systems. To provide wider applicability, we target HDFS clusters. HDFS is the distributed filesystem of Hadoop and the most prevalent framework of choice for distributed analytics. Despite its wide adoption, HDFS is still vulnerable to a variety of attacks and there exist scenarios where data integrity is at stake. For instance, when multiple untrusted parties have physical access to the cluster or in case of federated installations, HDFS cannot prevent malicious users from tampering the data.

§. The work has been done while Ioannis Mytilinis was with CSlab, NTUA.

To remedy this, we propose *Clouseau*, a system that integrates HDFS with the Ethereum blockchain to offer provable data integrity and tamper evidence at all times. In a nutshell, Clouseau uses blockchain as a secure coordinator, complementary to the existing HDFS Namenode. The on-chain metadata are accessed through smart contracts and are used to provide merkle proofs and prove integrity. For the sake of performance, we keep on-chain information to the bare minimum. Moreover, since we mainly use Ethereum as a trust delegator, exploiting its consensus mechanism, and are not interested in its financial implications, we rely on permissioned installations, where *gas* does not correspond to actual money.

In contrast to previous work [7] that assumes only Datanodes to be corrupted and builds a reputation-based system to penalize distrustful nodes, in Clouseau we make no such assumptions. Any party involved in the data storage process may have incentives to tamper the data.

Our contributions are summarized as follows:

- We propose a mechanism to guarantee verifiable integrity of distributed, shared data, leveraging merkle proofs and smart contracts. The efficiency of the mechanism is achieved by keeping the interaction with the blockchain to the bare minimum.
- We incorporate the proposed mechanism to the HDFS framework to provide trust in an untrusted environment, where any node may potentially be malicious. This is achieved at the cost of a configurable performance penalty. As we describe next, there are 2 tuning knobs that allow a user to explore the security-performance spectrum and choose the right configuration based on her needs.

2. Background

This section provides the necessary background for the understanding of the system. We first quickly recap the architecture of HDFS along with its deficiencies with respect to security and then discuss merkle proofs.

2.1. HDFS

HDFS comprises a master-slave distributed architecture, where the slaves (Datanodes) are responsible for storing and serving data, while the master (Namenode) coordinates client requests and maintenance operations, while doing the bookkeeping for data placement. Albeit the system provides a unified namespace, files are physically partitioned into blocks of configurable size (usually 128 MB) and are replicated over the network.

Writing a file requires the client to pack data into blocks. For each such block, the client contacts the Namenode and requests the address of a Datanode that can store the data. The Namenode uses a rack-aware scheme to select a Datanode and similarly to filesystems with POSIX semantics, it creates an *inode* that holds all the information of the specific block (e.g., location, file name, etc.). The HDFS inodes are maintained only by the Namenode and are stored in memory. Finally, the client, knowing the address of the Datanode, directly connects to it and transfers the block. The process of *reading a file* is similar: The client interacts with the Namenode, which provides the address of the Datanodes actually storing the file blocks, and the client can directly fetch the respective data blocks. The data blocks are accompanied by checksums, maintained by the responsible data nodes, which are used by the client to check the successful transfer of data.

Moreover, for health monitoring, Datanodes periodically send a *heartbeat* and a *block report* to the Namenode. The heartbeat is sent by default every 3 seconds and notifies that the specific Datanode is up and running. The block report is a more heavyweight operation that happens at larger time intervals (typically every 6 hours). During a block report, a Datanode sends information about all the blocks it maintains and a sanity check is performed at the Namenode's side. While the report is taking place, Namenode enters safe mode and does not accept client requests.

Evidently, apart from being a single point of failure, the Namenode is by default considered as trusted for the sound operation of the system. A malicious Namenode could alter inode information to intentionally hide parts of a file or add extra blocks to a file that never existed. Moreover, the Namenode could interfere in the block report process, falsely affirming tampered blocks as valid or, inversely, reporting valid blocks as corrupted. Furthermore, even under the presence of an honest Namenode, the existing block report mechanism cannot prevent malicious Datanodes from tampering the data, since the validation performed by the Namenode is based on the cross-check of simple metadata information such as size and generation timestamp, that do not concern the block's actual contents.

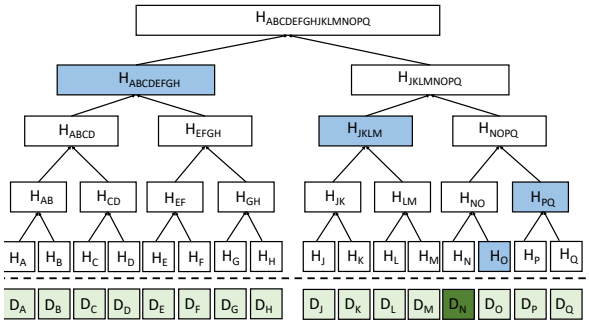

Figure 1. Merkle proof example

We argue that these are major shortcomings of current HDFS installations. Anyone (company, physical entity, etc.) that controls the Namenode or a set of Datanodes, can tamper files in a completely untraceable manner. This is exactly the problem that Clouseau aims to tackle.

2.2. Merkle proofs

In cryptography, a merkle tree is a tree where each leaf is the hash of a data block and each inner-node is the hash of its children. Figure 1 depicts an example. The green nodes below the dashed line correspond to data blocks, while the nodes above the line reflect the merkle tree.

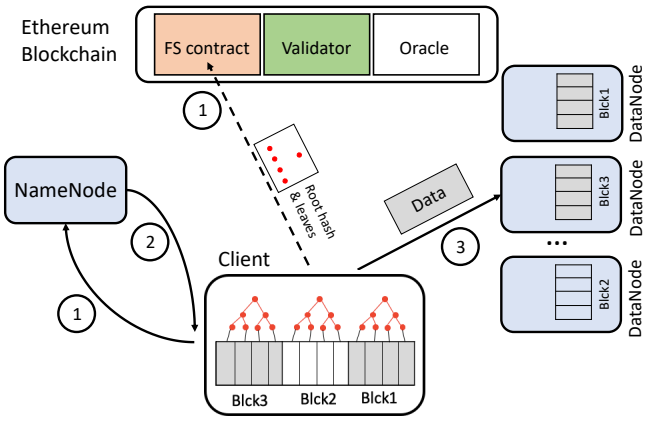

Figure 2. Writing a file

The reason why these trees are useful is because they can provide *inclusion* and *integrity* proofs. Inclusion guarantees that a data block has not been deleted and integrity guarantees that a data block has not been modified. These guarantees come in the form of merkle proofs.

Let us assume that Alice knows the root hash of a file and requests Bob to prove the inclusion and integrity of block D_N. Then, Bob hashes D_N and its sibling data block D_O and obtains H_N and H_O respectively. Then, he hashes H_N and H_O to get H_{NO} and so on until he computes the root hash. When the computation of hashes is done, Bob sends to Alice: (i) the data of the requested block (D_N) and (ii) the sibling nodes of the ones he computed at each level of the merkle tree (annotated with blue color in Figure 1). Having this information, Alice can reconstruct the path to the root and check whether the computed root hash equals the one she already has in her possession.

3. Architecture

Data integrity in Clouseau is verified with the use of merkle proofs through a set of Ethereum contracts. This section describes the architectural design choices, the integration between HDFS and Ethereum and how the synergy of the two creates a trusted environment for analytics.

Ethereum in Clouseau acts as a third-party auditor that tracks the state of uploaded files. In case a malicious subset of participating nodes, be it the Namenode and/or Datanode(s), tampers the data, the auditor detects the tampering and broadcasts a message with: (i) the identity of the dishonest party, (ii) the id of the corrupted block and (iii) a timestamp. To provide this auditing functionality, we have modified the read/write operations of HDFS, as well as the periodic block report procedure. Also, we had to equip each HDFS entity with an accompanying DApp with its own wallet and cryptographic key-pair. Finally, our protocol requires 3 more smart contracts that act as referees and are not associated with any Datanode (or the Namenode). These smart contracts have been uploaded on chain by the administrator of the system upon startup and their addresses were broadcasted to all members of the cluster.

HDFS write operation: The operation is illustrated in Figure 2. The client first splits the file to be uploaded into blocks as in the regular case. To be able to provide data integrity guarantees, we construct a merkle tree for each block. We further split the block into smaller chunks and follow the construction process described in Section 2.2. The selection of the chunk size of each merkle leaf determines the size of the merkle tree and provides a tunable knob between performance and security. Smaller chunks increase security at the cost of degraded performance.

After the client constructs the merkle tree, two actions are performed in parallel: (i) a request is sent to the Namenode and (ii) the root hash and the leaves of the merkle tree are stored on chain. For this purpose, we have created the *FS smart contract*, which is responsible for storing state and metadata that would normally reside in Namenode's memory. As a blockchain transaction can introduce significant latency and we wish to avoid a performance overhead in the critical path of HDFS I/O operations, this call happens in an asynchronous manner (signified by a dashed line).

HDFS read operation: We have modified the HDFS API to offer the flexibility of choosing once again between performance and security. The typical *get* operation of HDFS is still available for non-critical data. However, we also allow the client to opt for integrity validation upon a read request, as an extra sanity check apart from the periodic block report mechanism described in the following. During a secure HDFS *get* request, the client directly contacts the blockchain and retrieves a checksum for each block she reads. This way, we can guarantee integrity while making no assumptions on the Datanodes' honesty.

Block reports: In Clouseau, Datanodes send block reports not only to the Namenode but also to the third-party auditor, i.e., the blockchain. The initiation of a block report by a Datanode signifies the creation of k merkle proofs for the block. The k-value is a second knob of the system that trades security for availability: Higher k-values raise the probability to identify a potentially malicious action, but at the same time result in longer block reporting procedures, during which the Namenode does not accept client requests.

A question that naturally arises is who decides on the k specific chunks that are requested. The Datanode is precluded, since a malicious Datanode would always make convenient choices that avoid tampered data. The Namenode is not a wise choice either: A malicious Namenode could conspire with a Datanode and designate only pre-agreed proofs. Thus, we delegate the decision to the blockchain. This is the purpose of the *Oracle* contract. Given k, a Datanode and a timestamp, it generates k proof requests.

In order to maintain low communication costs, instead of returning the k proof requests per se, the Oracle sends back only a seed that can be used by a pseudo-random generator of the Datanode to generate a deterministic sequence of k chunk ids, i.e., the k proof requests. Thus, given a seed, the Datanode knows for which blocks it should provide proofs and the blockchain knows which proofs to expect.

Merkle proofs are constructed in the standard way and emitted to the *Validator* contract. Validator checks two things: (i) whether the proof is provided for the chunk for which it was requested and (ii) whether the proof is valid.

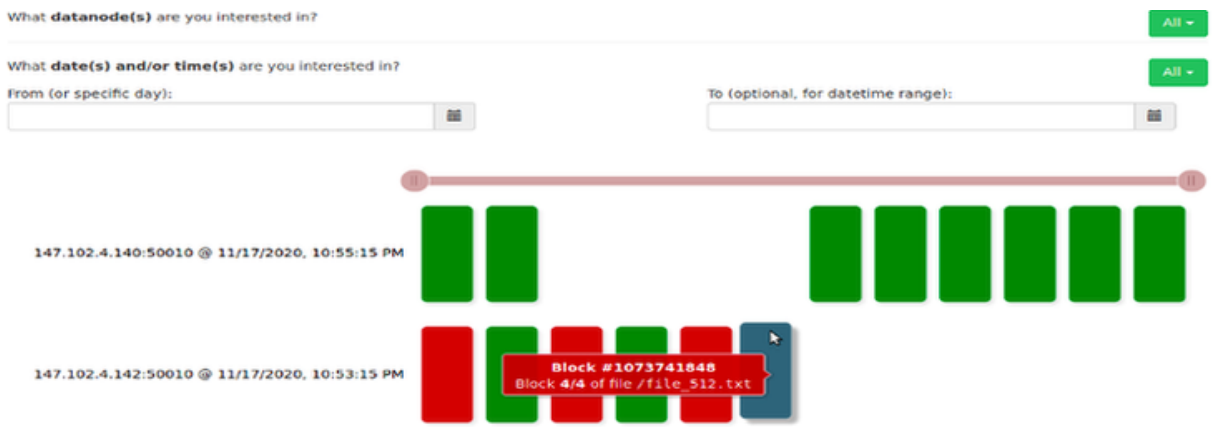

Figure 3. UI screenshot for block report

Algorithm 1: Merkle proof validation

input: timestamp t, Datanode address da, num of proofs k, homomorphic hash H

```
// Computations done at the Oracle
seed = oracle.generate(k, t, da);
send seed back to the Datanode through a
  blockchain event ;
// Datanode computations
for i in [1, k] do
    chunk_id = chunk_generator(seed).next();
    proof = merkle_proof(data[chunk_id]);
    send(chunk_id, merkle_proof);
// Validator computations
foreach (chunk_id, proof) do
    assert(fscontract.getLeaf(chunk_id) =
      proof.leaf);
    assert(validate_proof(proof) =
      fscontract.getRootHash(chunk_id));
```

For the former, Validator consults the leaves stored in the FS contract for the specific block, and for the latter it relies on the block's root hash. Algorithm 1 presents the whole validation process.

As the standard merkle proof validation procedure dictates, merkle proofs need to be accompanied by the raw data of the corresponding chunks/merkle leaves, to ensure the Namenode stores the actual data and not just the resulting merkle tree. Since this practice might violate privacy, alternative ways for data verification are examined, including zero-knowledge proofs. This is a subject of future work.

4. Demo Description

The demonstration will allow attendees to interact with Clouseau, offering them the opportunity to upload files, trigger block reports and finally inspect identified inconsistencies in the data. All these operations can be performed through an intuitive web user interface that visualizes health and integrity of data stored in the system.

For the ease of adoption, but also for the sake of seamless integration, we have extended the HDFS UI. The web pages traditionally used for browsing the filesystem now contain an additional health attribute that indicates the percentage of corrupted blocks as per the last report. Moreover, a new *Security Admin* page has been created. As it can be seen in the screenshot of Figure 3, we can use this dashboard to visually explore the proofs of the block reports and guide the exploration by various attributes such as the report dates, the Datanode of interest, the file of interest etc.

To facilitate block corruption for the demo purposes, we have created a web page that is exposed by each Datanode and permits to tamper data following different error distributions. Thus, the attendees will have the chance to corrupt data themselves and then trigger a block report to witness the ability of the system to detect malicious actions. The demonstration will also allow attendees to explore the trade-offs of the performance-security continuum in Clouseau. By selecting chunk sizes of various granularities and by requesting a variable number of proofs per block report, attendees will have a first-hand experience on the impact of Clouseau's tuning knobs.

5. Acknowledgement

This work has been supported by the European Commission in terms of the H2020 ELEGANT Project (957286).

References

[1] "Hadoop in secure mode," https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SecureMode.html, accessed: 2020-11-16.

[2] "Apache sentry," https://sentry.apache.org/, accessed: 2020-11-16.

[3] L. Allen, P. Antonopoulos, A. Arasu, J. Gehrke, J. Hammer, J. Hunter, R. Kaushik, D. Kossmann, J. Lee, R. Ramamurthy *et al.*, "Veritas: Shared verifiable databases and tables in the cloud," in *9th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2019.

[4] M. Zhang, Z. Xie, C. Yue, and Z. Zhong, "Spitz: a verifiable database system," *arXiv preprint arXiv:2008.09268*, 2020.

[5] F. M. Schuhknecht, A. Sharma, J. Dittrich, and D. Agrawal, "Chainifydb: How to blockchainify any data management system," *arXiv preprint arXiv:1912.04820*, 2019.

[6] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, 2016.

[7] S. M. Khan and K. W. Hamlen, "Hatman: Intra-cloud trust management for hadoop," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 494–501.