Cluster Computing
https://doi.org/10.1007/s10586-024-04646-6

File chunking towards on-chain storage: a blockchain-based data preservation framework

Muhammed Tmeizeh[1,2] · Carlos Rodríguez-Domínguez[2] · María Visitación Hurtado-Torres[2]

Received: 5 March 2024 / Revised: 14 June 2024 / Accepted: 15 June 2024

Abstract
The growing popularity of the most current wave of decentralized systems, powered by blockchain technology, which act as data vaults and preserve data, ensures that, once stored, it stays preserved, considered to be one of the most promising safe and immutable storage methods. The authors of this research suggest an on-chain storage framework that stores files inside blockchain transactions using file transforming, chunking, and encoding techniques. This study investigates the performance of on-chain file storage using a simulated network blockchain environment. Test files of varying sizes were deployed. Performance metrics, including consumed time in chunking, encoding, and distributing chunks among block transactions, were measured and analyzed. An analysis of the collected data was conducted to assess the framework's performance. The result showed that selecting the appropriate chunk size significantly influences the overall performance of the system. We also explored the implications of our findings and offered suggestions for improving performance within the framework.

Keywords On-chain · File chunking · Blockchain · Immutable storage

1 Introduction

One of the most important research topics in recent years has been the widespread adoption of the most recent breed of decentralized systems, powered by blockchain and decentralized file storage, that act as data vaults, protecting data to ensure that when it is stored, it remains unchanged (i.e., data is immutable) and can be used as secure storage, for instance, for archiving files or data, property rights, medical health records [1], and smart city data [2]. Some studies such as [3] examined open difficulties and unresolved problems in data security and the potential of integrating blockchain technology with many aspects, such as IoT and health data.

To ensure that an organization or business maintains its reputation and thrives, data security must be guaranteed. Data integrity is considered one of the main security principals. An IBM Cost of a Data Breach Report published in 2023 showed that data storage is one of the most targeted by frequent cyberattackers and has breaches that cost, on average, more than 12 million USD [4]. Significant consequences from data breaches could include financial losses, reputational harm, and legal obligations. Consequently, it is imperative that businesses implement strong safeguards to protect their data integrity. The integrity of electronic files is growing in significance at a time when electronic data is the foundation of many vital systems and activities, from financial records and transactions to medical records, legal documents, intellectual property, and scientific research. This paper aims to address the ongoing problems of fraud and data tampering by giving files the same level of immutability that usual blockchain records have. Immutability offers a strong base for preserving open

Carlos Rodríguez-Domínguez and María Visitación Hurtado-Torres contributed equally to this work.

✉ Muhammed Tmeizeh
mohammedt@paluniv.edu.ps; matps@correo.ugr.es

Carlos Rodríguez-Domínguez
carlosrodriguez@ugr.es

María Visitación Hurtado-Torres
mhurtado@ugr.es

[1] Faculty of Engineering and Information Technology, Palestine Ahliya University, Bethlehem 90907, West Bank, Palestine

[2] Department of Software Engineering, Higher Technical School of Computer and Telecommunications Engineering, University of Granada, 18071 Granada, Andalusia, Spain

Published online: 29 June 2024

and auditable documentation, safeguarding against malicious modifications, and enhancing trust in digital ecosystems.

The authors undertook a previous study [5] in this field, showing blockchain as a promising solution for secure data storage. That study showed that many proposals have been made to take advantage of the immutability feature of blockchain technology. A large number of those employ blockchain-connected external storage to save a hashed value that serves as a means of preventing the original saved data from being altered or forged [5]. Furthermore, other proposed works sought to save the data directly on the blockchain ledger; the metadata, which is very small in nature and includes things like gene names, light-weight sensor data, or even changes made to specific network nodes to enable the saving of data which is linked to blockchain ledger [5]. Our goal in this study is to present a framework that makes possible to store data directly into blockchain world-state ledger, which is regarded as an on-chain file storage solution. Moreover, data stored off-chain in Decentralized File Storage (DFS) connected to its hash value within the blockchain is not entirely safe since some DFS has security or immutability flaws [6]. Our previous survey study showed that the existing on-chain approaches remains limited by its inability to handle the full file storage inside the ledger, which is increasingly relevant in organization and business that seeks for tamper proof storage. This work introduces an innovative approach to extend the on-chain capabilities to adapt the file to be stored inside the ledger of the blockchain. Additionally, offering superior performance by optimizing both storage and retrieval speed addresses a critical gap in decentralized storage solutions.

Many blockchain-based solutions, such as off-chain approaches and non-blockchain-based solutions such as the Interplanetary File System (IPFS), introduce tamper-resistant storage, which aim to demonstrate the reliability that stored data has not changed [5], since such approaches serve as verification and checksum systems it cannot prevent participating nodes that store data fragments from updating or deleting it [6]. Linking data from outside the blockchain with its hash value inside the blockchain presents a hurdle when using the DFS with blockchain [7]. A centralized index is a problem for peer-to-peer decentralized storage [7]. However, in this paper, we aim to demonstrate with high reliability that data has not and also cannot be changed after being saved. As data and transactions that are saved inside the blockchain ledger are tamper-resistant, immutable and cannot be changed after being appended to the chain ledger, we aim to utilize this concept to save the file itself inside the distributed ledger. Moreover, a decentralized indexing system will also be provided within the blockchain. This approach empowers general-type file storage by providing a secure and efficient environment in terms of data integrity and immutability.

In this study we suggest an architecture that uses permissioned blockchain technology to allow file storage inside the blockchain ledger. This on-chain solution provides an immutable guarantee for the files since the whole file is being stored inside the ledger. Using a permissioned blockchain network such as Hyperledger Fabric, besides the access roles and authorizations offered, will facilitate file sharing between parties or organizations in a secure and immutable manner. The main contributions and benefits of the suggested framework are:

- providing a comprehensive on-chain solution for general-type file storage;
- reducing the processing load of saving data on the blockchain side by dividing tasks across the edge and core blockchain network;
- applying compression techniques lessens the amount of data in memory by shrinking the file, which optimizes ledger storage capacity and enhances storage effectiveness;
- leveraging the concept of storing files within a database as Binary Large Object (BLOB) to facilitate the storage of files within a blockchain ledger using a similar approach.

The remainder of this paper is organized as follows: Sect. 2 introduces an overview of related works by classifying, summarizing, illustrating related articles. Section 3 introduces an overview of the proposed framework while also providing a comprehensive background on relevant subjects in order to put the suggested framework into context. Section 4 shows testing of the proposed framework. Section 5 discusses in-depth the core elements of the framework, which also explain each component's presence within the framework and justify its utilization for each function. Finally, Sect. 6 concludes the paper and outlines some future work.

2 Related work

This section presents the related works of researchers that contributed to utilize blockchain as an immutable storage medium. Immutable storage using blockchain technology has been addressed by several approaches in recent years, two main classification approaches can be considered: on-chain and off-chain approaches.

Within the realm of off-chain, a diverse array of studies has been conducted. Each contributing in saving the data outside the blockchain while corresponding data are being stored in blockchain to verify the original off-chain data. For instance, Babu et al. [8] used MongoDB to store

electronic healthrecords (EHRs) while its hash value with some patient information will be stored in the blockchain. Yang et al. [9] proposed a solution to save data of the products into two parts, the first part called public data which will be saved in relation database where the private part of the data will be stored in the blockchain.

Another approaches in off-chain that use IPFS for instance, in [9] the authors use IPFS to save the data generated by IoTs in smart city applications then linked the data with Ethereum blockchain. Additional study by Mani et al. [10] uses a similar approach for saving data in IPFS which is connected to a Hyperledger blockchain that used as index in the form of key-value to reach the date in IPFS. Liu et al. [11] provided a way in his work to save images in IPFS and save the associated hash for the images inside the blockchain.

Furthermore, some studies accomplish the off-chain storage using cloud storage which is connected with blockchain. Chen et al. [12] used blockchain to index the cloud data as well as saving some metadata of the cloud records. additionally, Alrebdi et al. [13] used cloud storage with blockchain, the role of blockchain is to serve as access control and indexing system for the cloud data.

In a related study proposed by Pincheira et al. [14], which considered an off-chain solution, the research yielded a mechanism to store datasets as files within the distributed file system, with the metadata of this set being stored as a verification environment within the Ethereum public blockchain. This effort allows users to update their dataset while maintaining a fresh transaction within the blockchain to continuously audit the data life cycle, making the data more traceable. Testing was conducted utilizing IPFS, Swarm, and Git storage as the DFS on a range of file sizes, from 1 to 500 MB.

On the other hand, many studies conducted using on-chain data storage for instance, Sharma and Park [15] used blockchain to store a transactions of IoTs of smart city application in the form of hashed values which aimed to increase the security of data. Loss et al. introduce and integration between FIWARE and blockchain to store data in blocks, the authors used Ethereum in their work. As well Khalaf and Abdulsahib [16] introduced a traceable way of storage using blockchain, the study provides framework for saving the wireless network sensors broadcasting data as a transactions inside the blockchain.

Additionally, Arslan and Goker [17] introduced a new approach to store multimedia data inside blockchain node by making some modification on the miners behavior. Gürsoy et al. [18] store pharmacogenomics as a string inside the transactions associated with transaction ID in the form of key value pairs which is related to some of gene attribute.

Further research for Xie et al. [19] has proposed an on-chain solution to store IOT data for agricultural projects. An example of the collected data is humidity, temperature, GPS location, pressure, and so on. The data is meant to be saved inside the Ethereum blockchain as a transaction, and the transaction ID, which is a hash value, will be saved in an external traditional database with an identifier for each entity. This value will be used to retrieve data from the blockchain. Beside the main goal of storing data in immutable storage, the study concerns historical transactions for each entity. Additionally, [7] describes different methods-such as off-chain and on-chain solutions-for using blockchain technology for storage. Additionally, it illustrated the primary obstacles facing on-chain solutions, which can be summed up as follows: block size, quantity of transactions to be handled, gas cost per block, and locating a minor ready to handle such transactions.

As illustrated most approaches and studies are focused on off-chain approach, since it offers many benefits such as the ability to store a variety of data formats and files as well as the storage capacity. Whereas fewer papers delve into on-chain approach due to it's nature that poses some restrictions on the format on the type of data that can be saved in addition to some other factors which is related to the blockchain capacity. Our previous study [5] illustrated this point with more details. There are several limitations with on-chain data storage in terms of the kinds of data that can be stored. Saving general data formats like media files, documents, and data sets cannot be done directly in the ledger transaction due to the nature of the data that can be handled in the transaction, such as block metadata, timestamps, hashes, short text, and encoded binary data. Since the data structure of blockchain technology is primarily designed to store transactions of digital currencies within a connected chain of blocks. Our goal in this work is to establish a general framework for data files with any type of extension or format.

While the study [7] highlights the issue of fixed-size block examples such as the Bitcoin platform or even platforms that can adapt the change for the block size, we solved this by adapting block-less blockchain platforms. Another issue regarding saving the data that exceeds the capacity of a block is that we split the data in order to make it possible to distribute them among different blocks while it is easy to collect them based on the indexing criteria that we develop. Since the platform that we adopted is for enterprise use, an organization that wishes to deploy such a solution does not have to pay any gas costs since they share their infrastructure with other organizations that do the same in order to preserve their data.

Although [14] and other works had similar objectives, our work has a different approach, since we aim at saving the file in a single blockchain environment instead of

combining a blockchain with an external DFS. Likewise, [19] also aim at the on-chain idea, however, that study proposed the use of an external database as a source of data indexing, which is non-distributed and non-tamper-proof storage, and any loss of its content will leave the related blockchain data pointless and useless. On the other hand, our study does not rely on any external or integrated storage, and the indexing is totally saved inside the blockchain itself using integrated indexing chaincodes. While in our study the data will be distributed to all participating nodes who are enrolled in each private channel, some studies, like and [17], suggest on-chain approaches by changing the behavior of some nodes to store the data, but there is no guarantee that the data will be replicated to other nodes. Additionally, while [18] provides a paradigm for storing genetic data, it does not provide a universal file storage solution similar to what it is proposed in this paper.

3 Overview of the proposed framework

Our proposed framework is an on-chain storage framework, using consortium blockchain between members that want to store their files in an immutable storage by get the benefits of tamper proof characteristic that blockchain offers. Figure 1 shows the general structure of the proposed framework, in this framework any participating member or organization has nodes called peers which are configured to be invoked from an authorized clients using Application Programming Interface (API) and Software Development Kit (SDK). Authorized and authenticated clients or peers are eligible to store and retrieve files to and from the network.

In our framework, we use the Hyperledger blockchain Platform [20], which acts as a trusted infrastructure by authenticating any new member, organization, or node that wants to join the network. Besides, it's ability to offer a private channel between members or organizations that wish to make shared immutable file storage and need to impose their policy of committing or reading files to the network.

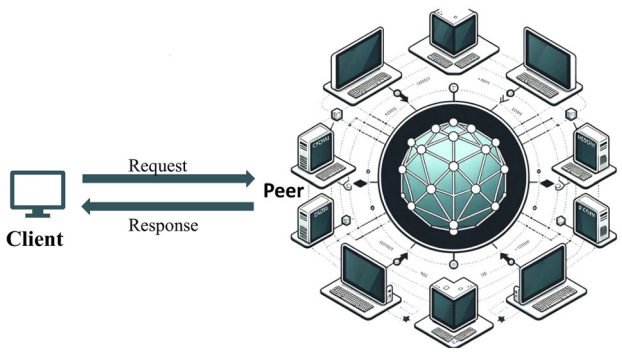

Fig. 1 General structure of the proposed framework

The client edge and the blockchain network share the workload in our framework. The file should be chunked and transformed by the client before being sent over the network. The data transformation and storage are handled by the core network, also known as the blockchain network. Further details regarding the framework are given in the subsections that follow.

3.1 Background

The background of the suggested work is presented in this section. Immutable data storage means that after storing the data, editing, altering, or deleting this data is not possible. In the real world, there's a need for such storage. For instance, medical trials, EHR, academic certificates, business contracts, and many other similar cases are all seeking such a solution. For instance, when someone buys a house, an ownership contract is issued, and if the same house is sold, a new contract is issued. The old contract should not be overwritten; instead, it should be appended to the existing one.

Hyperledger [20] is a general-purpose permissioned blockchain that gives peers the ability to execute smart contracts called chaincodes, which can be written using general-purpose programming languages such as Go, Java, and Node.js. It enables authenticated clients to send transactions to their peers. The transaction lifecycle goes through different entities: endorsing peers check the validity and correctness of the transaction; the ordering service uses the endorsement policy to order them chronologically into new blocks; then broadcasts the new state updates to peers and establishes consensus; finally, blocks are delivered and committed to the current state database for all peers on the channel.

Hyperledger consists of one channel or more; the channel is established between two organizations or more. Each channel has its peers. As illustrated in Fig. 2, we have a Hyperledger with three organizations and two channels: channel A is between organization 1 and organization 3, and channel B is between organization 2 and organization 3. Each channel has its own distributed ledger. Each peer who joins the network must have an identity issued via a membership service provider (MSP). As illustrated, an organization can be a member of one or more channels.

Applying our framework in blockchain with Hyperledger Fabric's modular design requires careful consideration of certain essential ecosystem modules. The goal of Fabric-CA is to issue and manage user IDs and certificates since any connection that is formed should be done so in a secure and legitimate manner. To demonstrate a real-world implementation case, any member or organization enrolled

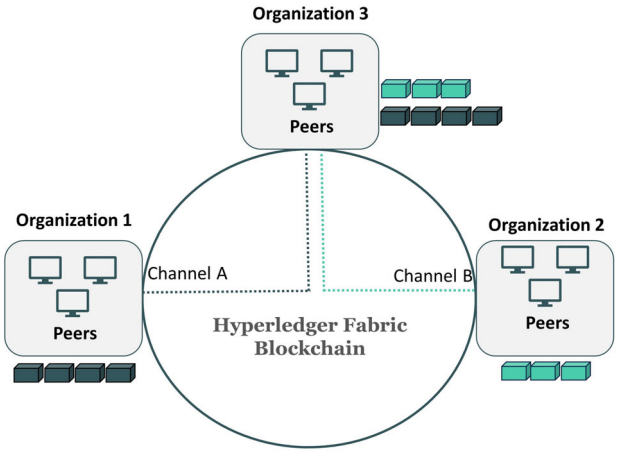

Fig. 2 Hyperledger channels example

in the framework would require a Fabric-CA in order to assign it the responsibility of issuing and managing cryptographic identities for peers, clients, and users. Moreover, a fabric peer module needs to be set up to host and execute chaincodes, which lets clients outside the blockchain network use the SDK to safely invoke and execute smart contracts. Additionally, the SDK is crucial since it provides a wide range of services and capabilities, such as event management, transaction processing APIs, cryptographic signature mechanisms, and logging frameworks. An endorsement policy configuration is needed to indicate how many peer nodes and members should accept the transaction, as the framework saves the data in blocks of transactions that are sent to peer nodes, who update the ledger accordingly. The actual use case and the members' agreement will dictate the configuration of this policy.

Hyperledger Fabric supports a number of consensus approaches that let network users decide on the order and validity of transactions. For instance, Kafka, RAFT, and Solo. The RAFT Consensus Protocol [21] is a simple and resource-efficient protocol that can be employed to run the framework. On the other hand, Kafka is not supported in the recent version [22]. However, since Solo relies on a single-node ordering mechanism, it is utilized in our testing network environments despite the fact that it is neither scalable nor fault-tolerant [22].

The suggested approach can offer technical remedies to deal with the aforementioned difficulties, such that if organizations wish to share immutable storage for their files, they can establish or join a blockchain network and create their own private channel. This channel will be used to save their data in a secure and tamper-proof manner.

3.2 Edge side

The edge or client side is responsible for preparing the file and sending it to the core blockchain network to save it directly inside the blocks of transactions. As illustrated from Fig. 3 the file will be zipped and converted to BLOB, then chunked with a fixed chunk size of 256 kilobytes, and each chunk will be converted to a Base64 string before the client sends the data to the peer in the core network. On the client side, each chunk is labeled with an index number before being sent to the blockchain core side. The process of indexing will be done using TypeScript application called LedgerLink on the edge side, as shown in Fig. 4; this code is also responsible for calling blockchain peers throughout the API using the SDK, which allows client applications to interact with a Fabric blockchain network.

Padding is not used whenever files are not precise multiples or less than the chunk size to prevent wasteful data duplication or modification. This guarantees that there are no extra padding bytes to optimize the consumed storage and that each chunk appropriately depicts the original data content. For example, if we have a 1000 KB file and want to split it into chunks of 256 KB, then the last chunk size will be 232 KB, occupying storage by full chunks will be $3 \times 256\,\text{KB} = 768\,\text{KB}$, and the last one will be $1000\,\text{KB} - 768\,\text{KB} = 232\,\text{KB}$.

As illustrated in Fig. 4, to save or retrieve a file, on the edge side, the FileHandler prepares the intended file to be saved by reducing its size, throwing out compression, converting it to BLOB, splitting it into chunks, and finally

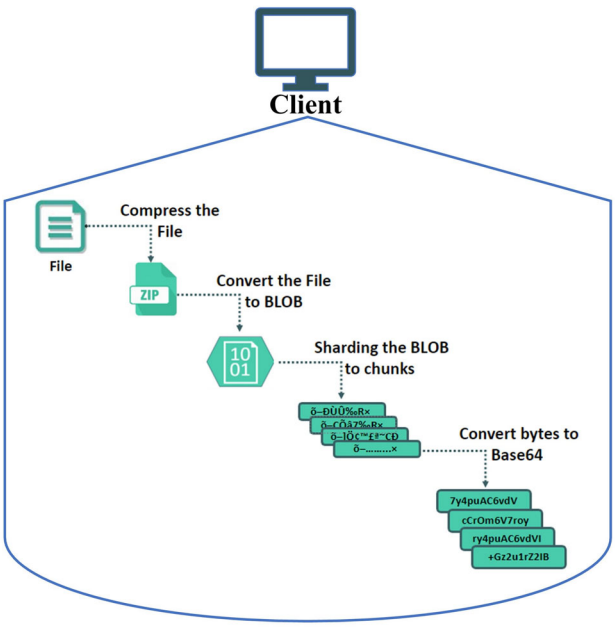

Fig. 3 Edge side: compressing, converting to BLOB, chunking, then transforming to Base64

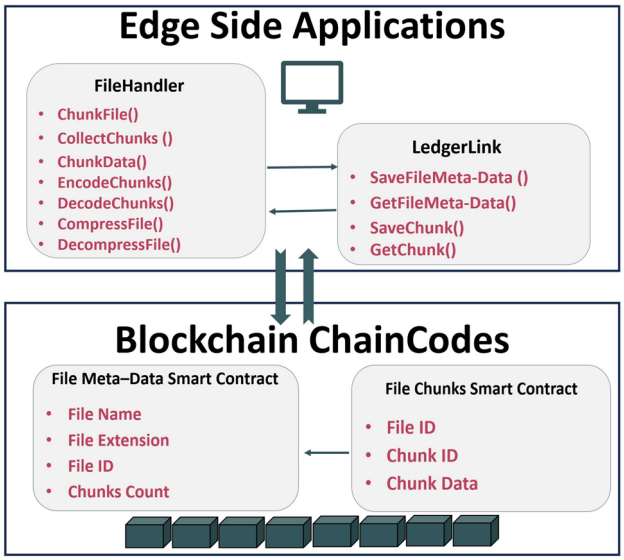

Fig. 4 Multi stage file processing interaction flow

encoding each chunk to bees64 format. When retrieving the file, the FileHandler will receive data from the LedgerLink app in the form of Base64 chunks. The FileHandler will decode the chunks, collect them into a BLOB, decompress them, and finally save them to retrieve the original file. The LedgerLink is responsible for establishing a secure connection with the blockchain in order to send or receive chunks. As well as save and retrieve the meta-data for files such as file ID, file name, chunk index number, file extension, and total count of chunks.

We utilize a compression strategy to minimize the file size; choosing the right algorithm is essential to utilizing the on-chain storage technique. Comparing different compression algorithms is done to find the best candidate for our work.

To make sure that every component of the file was intact during the compression and decompression procedures, we employed lossless compression in our framework. Lossless compression is crucial because of the design of our system, which protects the file's integrity and entirety. We can preserve the original file throughout the saving and retrieval stages by selecting this option, which ensures that no data is lost during the compression and decompression phases.

3.3 Blockchain side

On the core side of the blockchain network, the peer received a request to save file chunks in the blocks; as illustrated in Fig. 5, the chunks are received as a Base64 datatype. The chain code will transform the received chunk into a binary datatype and put it in an array of bytes, then send it to the transaction and, ultimately, to the blocks.

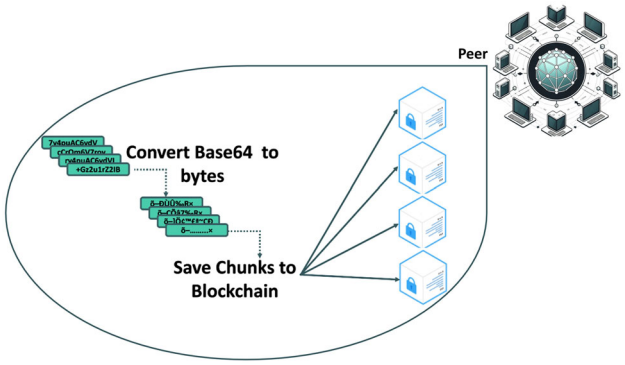

Fig. 5 Blockchain Peer Side: converting to binary data, then save the chunks in the transactions

3.4 File chunks indexing

Figure 6 illustrates what the used chaincode anticipates receiving from the client once file chunks are sent to the blockchain peer. Since each file will consist of many chunks, the formal arguments for the chaincode are the chunk in Base64 format, the hash value of the file identifying number, and the chunk number.

Every file will be splitted into one or more chunks, and each chunk will be assigned an index number, starting from zero during data processing in the LedgerLink app, which is located on the edge side as shown in Fig. 4. When saving and retrieving the file, the chunks will be retrieved in the order specified by this index number and other file meta-data that has been sent from the LedgerLink app to the File Meta-Data smart contract.

3.5 Saving and retrieving files

As instructed in Fig. 4, after preparing the file on the edge side from the FileHandler, the output will be sent to the

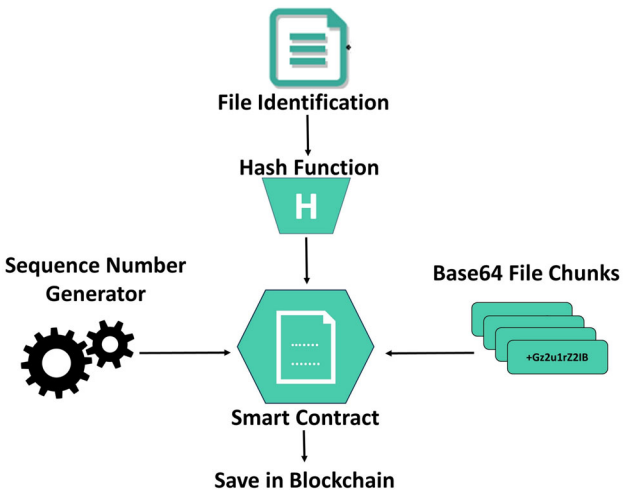

Fig. 6 File chunks indexing

LedgerLink, which in turn will read the received data and label each chunk with an index number, then send it to the blockchain side. The sent data will be divided into two categories of data. The chunks, their index order, and the file to which they belong make up the first category, which is the actual data of the file that will be sent to the file chunk smart contract. The file's meta-data, which includes the file type name, file ID, total number of chunks, and other details, falls into the second group, which will be sent to the File Meta-Data smart contract. The file name and extension will be used to construct the original file in the event that a legitimate party requests the file; the file id is the hash value of the file name using SHA256; the total number of chunks will be used for extracting the file.

However, in order to retrieve the file ID, chunk count, and other relevant information, the file reading process will be issued from the LedgerLink by making use of the information in a smart contract marked with File Meta-Data. The File Chunks smart contract will utilize the output to retrieve the chunks along with their corresponding index order, allowing the pieces to be arranged on the edge side and in a distant procedure. In order to create the desired file, the FileHandler will receive the file meat data and encoded chunks from the LedgerLink on the edge side.

4 Experiments and results

In this section, we present the outcomes of implementing our proposed framework for saving text files as a case study. The results provide insights into the effectiveness and performance of our framework in addressing the challenges of data storage and efficiency in decentralized environments.

First, we describe the experimental setup and methodology employed assess the functionality of the framework. The information gathered from our trials is then presented, including metrics for file size reduction, time spent chunking files, time needed to convert chunks to Base64 encoding, and transaction throughput on the blockchain network.

Following that, elements of the suggested framework have been deployed in an Oracle virtual machine (VM) running Ubuntu 22.04. 8 GB of RAM and four 1.80 GHz Intel (R) Core (TM) i7-8565U CPUs were set up in the virtual machine. The suggested system is then operated as a simulation environment on a test network running the Hyperledger Fabric-2.5 version. The simulation fabric network is implemented in Hyperledger Fabric and consists of two organizations, one peer node for each organization, a shared ledger among members, ordering service, endorser peer node, fabric certificate authorities through MSP, and chaincode.

The authenticated end-user application on the edge side interacts with blockchain peer chaincodes through the SDK to send data chunks to the ledger by connecting the endorsed peers when they need to store files in the ledger. The edge node application is divided into two sub-applications. The first is implemented in Python, which receives the file, compresses it, converts it to a BLOB, chunks the file, encodes each chunk to a Base64 string, and then saves each chunk in a distinct line in a text file. The second subpart is implemented using TypeScript, which takes the output text file of chunks and the hash code of the original file, then connects to the chaincode to send the chunks to be stored in the ledger.

Distributed File Systems dive deeply into selecting the optimal file chunk size; for instance, Swarm [23] uses a 4 kilobyte (KB) default chunk size, while IPFS [24] uses 256 KB. We conducted our studies for both sizes (4 and 256 KB) on the basis of that.

We ran two separate experiments with 12 randomly selected PDF files for each of the two chosen chunk sizes (4 KB and 256 KB). The majority of random files include only text or text with charts, with the exception of the five megabyte (MB) file, which is an electronic PDF book. Tables 1 and 2 present the measures and outcomes of the two experiments. As can be seen from the tables, the tests' largest file size was approximately 10.5 MB, while the smallest file size was just about 100 KB.

Figure 7 shows that it takes time to split the file into chunks using 4 KB and 256 KB chunk sizes. Figure 8 illustrates the time in seconds for the client application to transform the chunk of binary data into a Base64 string format, which will be used to send the data to the core network for the same chunk sizes used before.

As a key goal of our suggested framework is to minimize file size before storing it in the blockchain ledger, we accomplish this task by applying the Zstandard compression algorithm. Figure 9 shows the size of the original file, the file size after compression, and the size of the encoded chunks following conversion to Base64 string formats.

By sending the chunks from the TypeScript client code one after another throughout invoking API procedures of the chaincode within the peer of organization 1 in the simulation network, Fig. 10 illustrates how long it takes for both chunk sizes to save the file in the core network. Every file's duration, from transmitting the first chunk to committing the last, is measured in milliseconds and subsequently converted to seconds.

To assess the performance of the overall solution, an evaluation of the file retrieval efficiency and speed was

Table 1 File saving data summary: using a 256 KB chunk size

File size (bytes)	Compressed size (bytes)	Chunk count	Chunking time (s)	Encoded size (bytes)	Encoding time (s)	Save time (s)
102,602	77,985	1	0.0000	103,980	0.0003	2.103
266,603	187,461	1	0.0000	249,948	0.0010	2.135
359,075	253,832	1	0.0000	338,444	0.0006	0.155
422,552	196,232	1	0.0000	261,644	0.0008	0.124
512,088	311,796	2	0.0002	415,732	0.0033	2.196
829,048	741,722	3	0.0005	988,968	0.0016	2.102
1,052,352	881,284	4	0.0005	1,175,056	0.0021	2.480
1,664,464	466,725	2	0.0003	622,304	0.0010	0.243
1,821,766	1,280,137	5	0.0007	1,706,860	0.0028	0.609
4,293,938	3,354,594	13	0.0016	4,472,824	0.0084	1.466
5,842,628	5,402,908	21	0.0027	7,203,932	0.0132	4.659
11,081,517	1,694,115	7	0.0009	2,258,836	0.0044	2.798

Table 2 File saving data summary: using a 4 KB chunk size

File size (bytes)	Compressed size (bytes)	Chunk count	Chunking time (s)	Encoded size (bytes)	Encoding time (s)	Save time (s)
102,602	77,985	20	0.0001	104,032	0.0001	40.805
266,603	187,461	46	0.0002	250,068	0.0005	93.688
359,075	253,832	62	0.0002	338,608	0.0011	126.077
422,552	196,232	48	0.0002	261,768	0.0005	97.605
512,088	311,796	77	0.0003	415,932	0.0014	156.531
829,048	741,722	182	0.0005	989,448	0.0020	370.133
1,052,352	881,284	216	0.0006	1,175,620	0.0023	438.965
1,664,464	466,725	114	0.0004	622,604	0.0021	231.815
1,821,766	1,280,137	313	0.0008	1,707,684	0.0034	636.022
4,293,938	3,354,594	819	0.0019	4,474,976	0.0099	1664.697
5,842,628	5,402,908	1320	0.0030	7,207,396	0.0144	2679.468
11,081,517	1,694,115	414	0.0010	2,259,924	0.0045	840.943

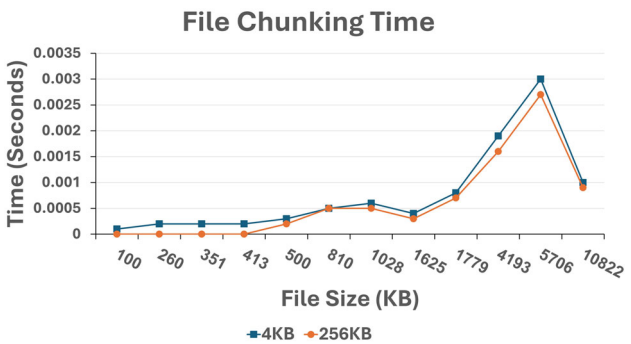

Fig. 7 Time consumed to chunk the file using 4 KB and 256 KB chunk sizes

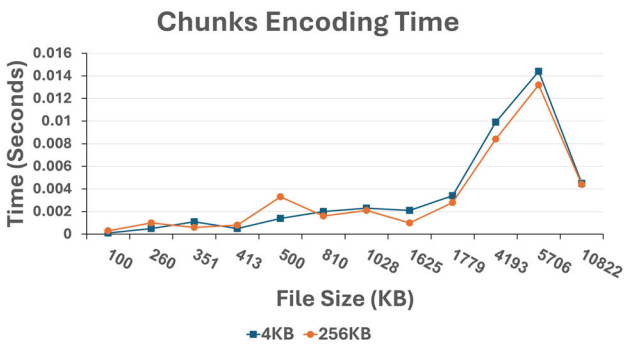

Fig. 8 Time consumed to encode the chunks to Base64 using 4 KB and 256 KB chunk sizes

conducted. Table 3 illustrates the consumed time in seconds for fetching data from the blockchain, decoding, decompressing, and saving that retrieved file. Given that the best practice and recommended chunk size was 256 KB in the previous subsection, the retrieval experiment was conducted on the same chunk size. In this experiment, the same 12 files that were previously saved were retrieved.

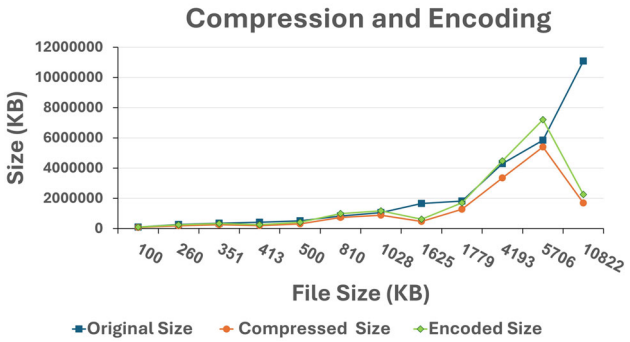

Fig. 9 File sizes before and after compression and encoding

Fig. 10 File-saving time in the core blockchain network

5 Discussion

5.1 Blockchain platform

Several blockchain platforms can be used to implement the suggested architecture. To determine which blockchain would be best for our job, we compared the Multichain and Hyperledger Fabric blockchains in Table 4. Transactions per second (TPS) are supported for both platforms; however, proper configuration and tweaking are mostly required. Hyperledger is an outstanding match for our work since it supports parallel transaction processing, which allows us to have a higher throughput in terms of TPS, and it offers a blockless design in terms of block size, also, supporting role-based access control (RBAC) fits our proposed work since enables organizations to grant peers access to the files based on their roles. This approach allows for the precise management of permissions and privileges. The comparison conducted through a review of Multichain white papers [20, 25].

5.2 File chunking

In the proposed work the file will be divided into segments or chunks in the edge side. Chunking module can be either fixed size or variable size. In our framework the we use fixed size chunking. Choosing the chunk size is an important issue since (i) choosing the optimal chunk size is a critical factor in effective chunk distribution; (ii) furthermore, the performance and reducing latency in the core blockchain network while transforming and saving the chunks; (iii) It is also a key consideration in optimizing bandwidth usage; (iv) Additionally, it has a direct relationship with consistency and integrity in the chunk distribution process in the public ledger.

Table 3 Retrieving file data summary: using a 256 KB chunk size

Size (bytes)	Fetching file from blockchain	Decode time (s)	Decompress time (s)	Save time (s)
102,602	0.057	0.0003	0.0004	0.0012
266,603	0.071	0.0006	0.0002	0.0006
359,075	0.072	0.0009	0.0003	0.0006
422,552	0.066	0.0006	0.0004	0.0012
512,088	0.075	0.0012	0.0005	0.0016
829,048	0.115	0.0041	0.0007	0.0021
1,052,352	0.122	0.0032	0.0013	0.0022
1,664,464	0.088	0.0016	0.0016	0.0035
1,821,766	0.162	0.0051	0.0016	0.0021
4,293,938	0.345	0.0116	0.0032	0.0048
5,842,628	0.518	0.0196	0.0060	0.0093
11,081,517	0.199	0.0061	0.0066	0.0207

Table 4 Multichain and hyperledger fabric comparison

Feature name	Multichain	Hyperledger
Scalability	Less	More scalable
Smart contracts	Yes	Yes
Code power	Simple scripting language	More powerful and flexible
Authorization	Permissioned blockchain	Permissioned blockchain (RBAC)
Block capacity	Default block size 2 MB	Blockless architecture
Privacy	Greater control using private chains	Fine-grained control over data access
Appending new block	Traditional way	Support parallel transaction processing

The chart analysis in Fig. 7 reveals that using 256 KB chunk size slightly outperformed 4 KB chunk size, albeit demonstrating a degree of symmetry or parity in many aspects.

5.3 Chunks encoding

The Base64 index table is used to convert the binary format that is fed into it into printable ASCII characters. It is possible to utilize many encoding algorithms to encode the file. Choosing the optimal algorithm for our task based on the highest performing methods. In terms of speed and performance, Base64 beats other common algorithms, including UUEncode, XXEncode, USR, BINHEX, BTOA, BOO, and Quoted-Printable [26]. It also performs well on a variety of processor types and includes standard libraries for well-known programming languages including JavaScript, Go, Swift, PHP, Python, C#, and Java [27]. Thus, Base64 is an excellent fit for our framework since it allows us to use our framework on a variety of computers and programming environments, especially on the edge.

We encode the data to Base64 before transmitting it to the blockchain after transforming the file to binary format. Sending raw binary data has some drawbacks compared to using Base64, as a result of which our suggested work is compatible with blockchain platforms that allow text-based protocols and formats. When transmitting data over an unencrypted connection, Base64 encoding additionally helps in avoiding data modification and interception. Additionally, because JSON payloads and URLs support text data, it is more convenient to communicate data to an API using them. However, this requires adding certain overheads to the framework, such as more processing and about a 33% increase in the size of the original data. The framework's last phase will store the data as binary rather than Base64.

Performance is an important consideration; therefore, we examine the time spent on the edge side of the encoding process, considering the fact that using the compression approach helps minimize the additional file size that results from encoding the file. The encoding time, which takes about 3.4 ms on average for the different chunk sizes we use in our tests for all files enrolled in the testing, is sufficient for both chunk size scenarios, as Fig. 8 illustrates.

5.4 File compression

An on-chain file saving is the goal of our endeavor. The ledger will eventually get larger because the files are kept directly on it. One useful method to take advantage of the blockchain's storage capacity is to compress files. We decide to compress the files on the client or edge side in order to minimize their size before saving them. In this study, the selection of an acceptable compression technique is primarily dependent on a few features, like minimizing the file size and maintaining the original compressed file without losing any of its contents.

Data compression algorithms come in two categories: "lossless" and "lossy". Text, strings, and programs can be compressed using lossless data compression techniques, while images, videos, and audio files can be compressed using lossy compression algorithms. We select lossless algorithms depending on the requirements indicated before.

Since our proposed framework is not meant to save a certain file type, selecting the optimal compression method will depend on the use case for the desired file type to store. Table 5 provides a list of possible techniques that can be used to compress files in the Portable Document Format (PDF), for instance. Since our goal is to minimize the size

Table 5 Examples of lossless compression algorithms

Algorithm	Mainly targeted file
Run length encoding	Simple graphic
Shannon-Fano coding	General-purpose
Zstandard	General-purpose
Lempel–Ziv–Welch (LZW)	General-purpose
Huffman coding	Text data
Arithmetic encoding	Text data
BZIP2	Text data
GZIP	General-purpose
PPM	General-purpose

of the ledger, the compression ratio is the basis for choosing the best algorithm. Based on the results of the RasterLite2 benchmark [28], Zstandard could be one of the best options for our chosen case, which is text files.

The sizes of the original file, the compressed file, and the total size of encoded chunks in the file are all demonstrated in Fig. 9. The figure shows that, despite the encoding process increasing the quantity of data, the encoding size for all used files is smaller than the original file size. This is due to the compression process that takes place prior to the encoding process.

5.5 Optimal chunk size selection

Based on the aforementioned results, we infer that there is a little variation in the various performance measures that we ran while employing a 4 KB or 256 KB chunk size on the edge side. Conversely, the choice of a suitable chunk size is dependent on optimizing the file's performance efficiency during the saving process on chunks in the block transactions. According to the results, it took less time to save the file in blocks while utilizing a 256 KB chunk size as opposed to a 4 KB chunk size. According to Fig. 10, saving a file in a block with a 256 KB chunk size often takes 1.75 s, but a file with a 4 KB chunk size typically takes around 614 s. This outcome derives from the concept that decreasing the chunk count is preferable to decreasing the number of calls to the chaincode's API.

5.6 File chunk retrieval

Figure 11 presents an examination of the retrieval time, or the amount of time in seconds needed to read files, using the data from Table 3, emphasizing significant trends and consequences. The data presented clearly show that the matching fetching or retrieving time and the compressed file size have a positive relationship. The fetching time grows progressively with increasing file size when the original file size is shown against the fetching time. This tendency indicates increased processing overhead, as the

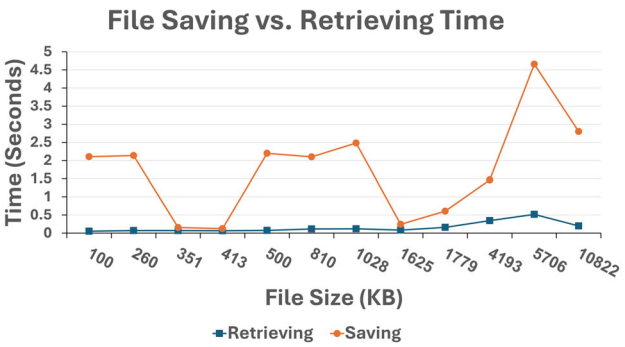

Fig. 12 File saving vs. retrieving time to/from the core blockchain network

increase in file chunks will take longer to retrieve. For instance, a file with a size of 5706 KB takes longer than a file with a size of 10,822 KB. This is because the number of chunks for the first one is 21, and the latest is 7 chunks, as shown in Table 1. The reason behind the increase in chunk numbers is that after compressing the two files, the results show notable shrinkage in the size of the latest one due to its content, which has fewer images and figures inside, and more text, which can be compressed with a higher ratio.

A comparison of the saving and retrieval times is presented in Fig. 12, which indicates a notable disparity between the two procedures. Retrieval is far superior to time saving, as shown. This finding is consistent with the observation that the frequency of reading operations relative to saving operations significantly reduces the value of saving time as opposed to retrieving time. While storing activities for any file usually only happen once, reading actions are carried out repeatedly, possibly on a massive scale. As a result, retrieval time optimization is more crucial for maintaining effective and responsive system performance because it affects both the user experience and system responsiveness when regular file access activities are performed. In contrast, although optimizing the savings of time is still important for overall system performance, it is not as important because it happens less frequently than retrieving.

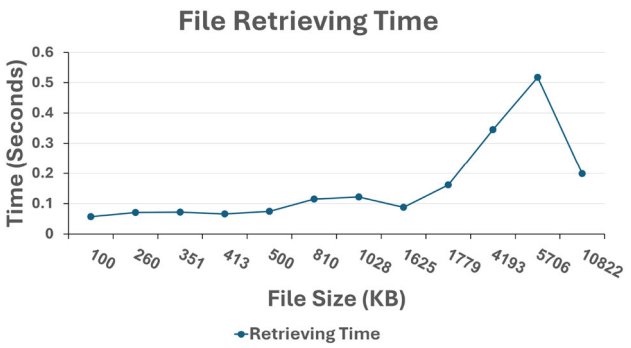

Fig. 11 File retrieving time from the core blockchain network

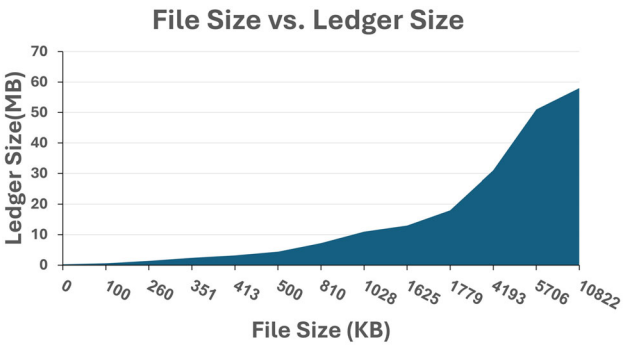

Fig. 13 File size vs. ledger size: analyzing relationship

5.7 Ledger data size

Figure 13, which depicts the relationship between file and ledger size, highlights another major challenge with ledger size. As we've previously discussed, a crucial component in many of our framework's disciplines is the quantity of chunks for each file. The figure shows how the ledger size is influenced by the final file data result following file processing, so the number of chunks will have an effect on the ledger size regardless of the original file size. As shown in the figure, the file with the size of 5706 KB, which has the most chunks, is the source of the notable increase in the ledger size, as seen in the figure. The authors will conduct a subsequent study on a real blockchain network to determine the ideal chunk size in a real-world setting, considering the bandwidth, core network performance, and ledger size optimization.

We conducted an experiment using 256 KB optimal chunk size to save the files utilized in our earlier experiments without compressing them in order to evaluate the effectiveness of applying compression techniques and their impacts on the ledger size. Figure 14 shows the ledger size in both cases of storing compressed and non-compressed files. As illustrated in the figure, there is a significant change in the ledger size when using non-compressed files. In summary, after saving all files used in the experiment, the ledger size in the non-compressed files scenario is about 105 MB, whereas the ledger size in the compressed files scenario is around 55 MB, which shows a significant reduction in the ledger size while saving the same data in both cases. This experiment highlights the importance of reducing file size using compression techniques suggested in our framework.

5.8 Time performance analysis

Although many performance measurements were covered in the previous sections, this section presents a time analysis to process the files in terms of compression, decompression, coding, decoding, and saving the retrieved file.

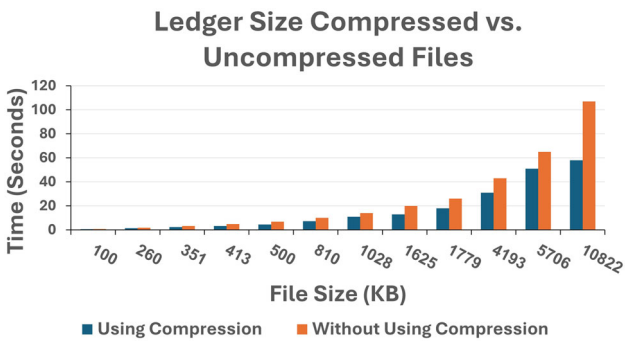

Fig. 14 Ledger size compressed vs. uncompressed files

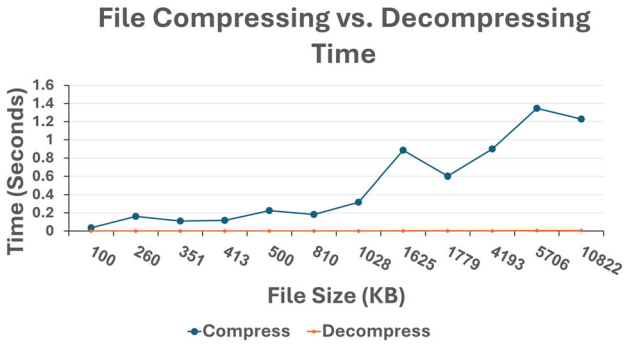

Fig. 15 File compressing vs. decompressing time

Figure 15 shows the consumed time for compression and decompression, which depicts that the average for compressing 12 sample files is 0.52 s, while the average for decompressing less than 0.002 s. These findings support our framework objectives, which deliberately prioritized effective reading above writing operations.

We examined the effectiveness of utilizing compression techniques in our suggested framework by analyzing the results of the previous experiment, which used a 256 KB chunk size to save the files without compressing them. This allowed us to determine the significance of using compression in the framework in terms of the time consumed for saving and retrieving data from the ledger. Table 6 displays the results for saving and retrieving the data. When saving files without compression, the average time in seconds is 4.89, whereas the time in seconds for saving compressed files is 1.61. Moreover, retrieving the compressed files takes an average of 0.16 s, whereas retrieving the same files without compression takes around 0.27 s. The time required to save each file to the ledger with and without file compression is displayed in Fig. 16, whereas the time required to retrieve data from the ledger with and without file compression is displayed in Fig. 17. The findings indicate that the time required for both processes increases as the amount of data increases without compression, suggesting that compression improves performance for both file saving and retrieval. Due to this, the 0.52 and 0.002 s required average for compression and decompression, respectively, are seen as acceptable for such operations.

Based on the optimal selected chunk size, which is 256 KB, Fig. 18 shows the coding and encoding time needed for the entire chunks of each file. The average time is less than 0.003 s, while the average for decoding is 0.005 s, which is also considered to be negligible. No experiment was carried out without encoding since it is an essential component of the suggested framework to avoid any compatibility issues as not all APIs or clients might support data in binary format.

Table 6 Time performance: saving and retrieving data with and without compression

File size	Using compression		Without using compression	
	Time to save (s)	Time to retrieve (s)	Time to save (s)	Time to retrieve (s)
102,602	2.103	0.057	4.158	0.064
266,603	2.135	0.071	4.23	0.072
359,075	0.155	0.072	4.246	0.076
422,552	0.124	0.066	4.34	0.084
512,088	2.196	0.075	2.327	0.09
829,048	0.382	0.115	4.496	0.12
1,052,352	2.48	0.122	4.576	0.15
1,664,464	0.243	0.088	4.841	0.249
1,821,766	0.609	0.162	2.917	0.202
4,293,938	1.466	0.345	6.092	0.442
5,842,628	4.659	0.518	6.714	0.56
11,081,517	2.798	0.199	9.767	1.119

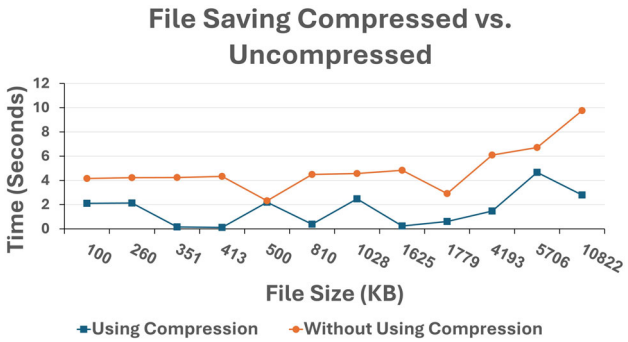

Fig. 16 File saving compressed vs. uncompressed

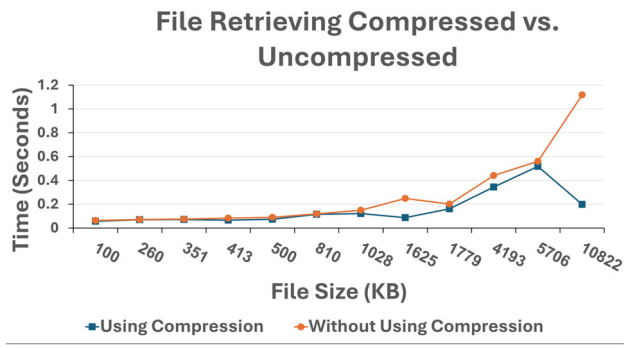

Fig. 17 File retrieving compressed vs. uncompressed

Finally, the last column in Table 3 shows the time needed to reconstruct and save the file in its final stage to appear ready for use. As illustrated in the table, the time needed to reconstruct the file is negligible and falls well within the expected performance parameters.

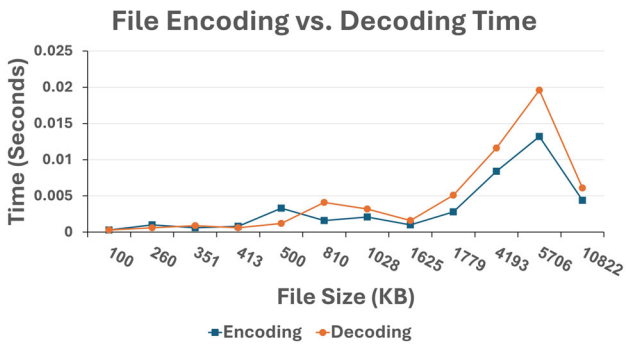

Fig. 18 File encoding vs. decoding time

5.9 Performance comparison

In this section, we are going to compare the results of our work with similar approaches that used on-chain solutions, taking into account that they're not totally identical approaches, and have been tested in different environments. For example, testing results in research [19] demonstrated that the proposed system can store around 300 sensor data per second, whereas reading the same data from the system takes a few seconds. Assuming that each sensor's data type is double, then 300×8 bytes a total of 2400 bytes are required. Returning to our suggested framework, 100.2 KB is the minimum file size that is used, which needs about 2.1 s to be saved to the blockchain, and retrieving it will take about 0.057 s.

The datasets used by the authors of [15] ranged in size from 1 to 8 MB. The average time to store data is 3.9 s. We assumed a mean data size of 3.8 MB in order to compare it with a comparable size in our experiments, based on the data sets employed in their study. On the other hand, our proposed framework takes less time (around 1.5 s for 4.1 MB).

Table 7 Performance comparison

Approach	Operation	Approach test data size	Approach time (s)	Our relevant data size	Our proposed framework time (s)
[19]	Save data	2400 B	1	100 KB	2.1
[15]	Save data	3.8 MB	3.8	4.1 MB	1.5
[18]	Save data	9.2 MB	3.4	10.57 MB	2.8
[19]	Retrieve data	2400 B	Few seconds	100 KB	0.057

The authors of [18] employ gene data entries that are stored using an Ethereum smart contract. The size of each 1000 entries is around 0.92 MB; in their work, 10,000 entries, or 9.2 MB, takes about 3.4 s; in our experiments, the best relevant size to compare with is 10.57 MB, which takes 2.8 s.

Considering that the previous works did not offer many experimental findings for data retrieval, Table 7 compares the time required for storing and retrieving data between our proposed work and the relevant works. In this study, we experimented with different file sizes and compared them with those used in comparable publications. However, because they are similar, some sizes can be compared.

Table 7 illustrates that our results outperform those reported in comparable studies with respect to the amount of time spent on data retrieval and storage. There are multiple reasons for this improvement. The data were first compressed to reduce their original size and then saved as raw binary data, which has superior performance and storage efficiency. Furthermore, certain preparatory activities, including chunking and data indexing, are carried out on the edge.

6 Conclusion

In this study, we set out to investigate the potential and performance of on-chain file storage in a blockchain ledger. The proposed framework reveals that blockchain offers a promising solution for organizations that seek an immutable storage solution with the power of authentication and authorization that the Hyperledger Fabric blockchain offers. Despite some limitations and constraints in terms of consumed time, performance, and ledger size to deploy the framework, the overall performance demonstrates its potential as a viable alternative to traditional centralized storage systems. In addition to evaluating the performance of our framework, we also shed light on the critical importance of selecting the appropriate chunk size, and compression techniques for efficient file storage.

While our study provides valuable insights into utilizing on-chain solutions, it is not without limitations. Future research efforts should focus on addressing these limitations and exploring opportunities for further optimization and enhancement of file storage solutions, such as investigating the effects of redundancy of storage, ledger capacity, encoding and appropriate file indexing.

Finally, our research highlights the importance of blockchain technology in tackling the changing problems associated with data storage. Organizations can explore new avenues for secure, effective, and unchangeable data storage solutions by adopting secure and decentralized principles.

Author contributions Each author contributed to the conception and design of the study, acquisition, analysis, and interpretation of data, drafting the manuscript, and revising it critically for important intellectual content. All authors have read and approved the final version of the manuscript and agree to be accountable for all aspects of the work. Muhammed Tmeizeh: conceptualization, methodology, formal analysis, writing—original draft, writing—review and editing. Carlos Rodriguez-Dominguez: supervision, project administration, writing—review and editing María Visitación Hurtado: supervision, project administration, writing—review and editing.

Funding Funding for open access publishing: Universidad de Granada/CBUA.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Shahnaz, A., Qamar, U., Khalid, A.: Using blockchain for electronic health records. IEEE Access **7**, 147782–147795 (2019). https://doi.org/10.1109/ACCESS.2019.2946373
2. Loss, S., Singh, H.P., Cacho, N., Lopes, F.: Using FIWARE and blockchain in smart cities solutions. Clust. Comput. **26**(4), 2115–2128 (2023)
3. Popoola, O., Rodrigues, M., Marchang, J., Shenfield, A., Ikpehia, A., Popoola, J.: A critical literature review of security and privacy in smart home healthcare schemes adopting IoT & blockchain: problems, challenges and solutions. Blockchain: Res. Appl. 100178 (2023). https://doi.org/10.1016/j.bcra.2023.100178
4. Gaia-GIS: Ibm security (2023). https://www.ibm.com/reports/data-breach. Accessed 2 May 2024
5. Tmeizeh, M., Rodríguez-Domínguez, C., Hurtado-Torres, M.V.: A survey of decentralized storage and decentralized database in blockchain-based proposed systems: potentials and limitations. In: International Congress on Blockchain and Applications, pp. 204–213. Springer, Berlin (2023)
6. Casino, F., Politou, E., Alepis, E., Patsakis, C.: Immutability and decentralized storage: an analysis of emerging threats. IEEE Access **8**, 4737–4744 (2019)
7. Hepp, T., Sharinghousen, M., Ehret, P., Schoenhals, A., Gipp, B.: On-chain vs. off-chain storage for supply- and blockchain integration. Inf. Technol. (2018). https://doi.org/10.1515/itit-2018-0019
8. Babu, E.S., Yadav, B.R.N., Nikhath, A.K., Nayak, S.R., Alnumay, W.: MediBlocks: secure exchanging of electronic health records (EHRs) using trust-based blockchain network with privacy concerns. Clust. Comput. **26**(4), 2217–2244 (2023)
9. Yang, X., Li, M., Yu, H., Wang, M., Xu, D., Sun, C.: A trusted blockchain-based traceability system for fruit and vegetable agricultural products. IEEE Access **9**, 36282–36293 (2021). https://doi.org/10.1109/ACCESS.2021.3062845
10. Mani, V., Manickam, P., Alotaibi, Y., Alghamdi, S., Khalaf, O.I.: Hyperledger healthchain: patient-centric IPFS-based storage of health records. Electronics **10**(23), 3003 (2021)
11. Liu, F., Yang, C., Yang, J., Kong, D., Zhou, A., Qi, J., Li, Z.: A hybrid with distributed pooling blockchain protocol for image storage. Sci. Rep. **12**(1), 3457 (2022)
12. Chen, Y., Ding, S., Xu, Z., Zheng, H., Yang, S.: Blockchain-based medical records secure storage and medical service framework. J. Med. Syst. **43**, 1–9 (2019)
13. Alrebdi, N., Alabdulatif, A., Iwendi, C., Lian, Z.: SVBE: searchable and verifiable blockchain-based electronic medical records system. Sci. Rep. **12**(1), 266 (2022)
14. Pincheira, M., Donini, E., Vecchio, M., Kanhere, S.: A decentralized architecture for trusted dataset sharing using smart contracts and distributed storage. Sensors **22**(23), 9118 (2022)
15. Sharma, P.K., Park, J.H.: Blockchain based hybrid network architecture for the smart city. Future Gener. Comput. Syst. **86**, 650–655 (2018). https://doi.org/10.1016/j.future.2018.04.060. https://www.sciencedirect.com/science/article/pii/S0167739X1830431X
16. Khalaf, O.I., Abdulsahib, G.M.: Optimized dynamic storage of data (ODSD) in IoT based on blockchain for wireless sensor networks. Peer-to-Peer Netw. Appl. **14**, 2858–2873 (2021)
17. Arslan, S.S., Goker, T.: Compress-store on blockchain: a decentralized data processing and immutable storage for multimedia streaming. Clust. Comput. **25**(3), 1957–1968 (2022)
18. Gürsoy, G., Brannon, C.M., Gerstein, M.: Using Ethereum blockchain to store and query pharmacogenomics data via smart contracts. BMC Med. Genom. **13**(1), 1–11 (2020)
19. Xie, C., Sun, Y., Luo, H.: Secured data storage scheme based on block chain for agricultural products tracking. In: 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), pp. 45–50 (2017). https://doi.org/10.1109/BIGCOM.2017.43
20. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, pp. 1–15. USENIX Association, Philadelphia, PA (2018)
21. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference (USENIX ATC 14), pp. 305–319. Association for Computing Machinery, New York, NY (2014)
22. Fabric, H.: Ordering service (2024). https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering_service.html. Accessed 5 May 2024
23. Trón, V.: The book of swarm: storage and communication infrastructure for self-sovereign digital society back-end stack for the decentralised web. V1. 0 pre-Release **7** (2020)
24. Benet, J.: IPFS-content addressed, versioned, P2P file system (2014). arXiv preprint. arXiv:1407.3561. Accessed 26 Feb 2024
25. MultiChain: Multichain private blockchain—white paper. https://www.multichain.com/download/MultiChain-White-Paper.pdf. Accessed 15 Feb 2024
26. Mustapa, M., Taliang, A., Iskandar, A., et al.: Comparison of encoding and decoding methods for binary files. J. Phys.: Conf. Ser. **1364**, 012024 (2019)
27. Muła, W., Lemire, D.: Faster Base64 encoding and decoding using AVX2 instructions. ACM Trans. Web (2018). https://doi.org/10.1145/3132709
28. Gaia-GIS: Benchmarks (2019 update) (2019). https://www.gaia-gis.it/fossil/librasterlite2/wiki?name=benchmarks+(2019+update). Accessed 25 Feb 2024

Muhammed Tmeizeh is a Ph.D. candidate at the University of Granada, Spain. He obtained his Master's degree in Informatics and Bachelor's degree in Computer Science from Palestine Polytechnic University. He is a researcher specializing in blockchain-based systems and a lecturer at Palestine Ahliya University, where he also serves as the Dean of Admission and Registration. Additionally, he is a team leader for many software development projects at Palestine Ahliya University. He is a reviewer for two prestigious international conferences. His work is focused on advancing the field of blockchain technology through innovative research and practical applications.

Carlos Rodríguez-Domínguez is an Associate Professor at the University of Granada, Spain, where he also obtained his Ph.D. in Computer Sciences and Master Degree in Software Development. He is also a co-founder and Chief Technology Officer at Everyware Technologies since 2012. He is a member of the Modeling & Development of Advanced Software Systems (MYDASS) research group. He participates as a guest editor of high prestige journals and as a member of several organizing and program committees of multiple conferences, workshops and scientific journals. He has participated as a developer and researcher in more than 20 R&D&I projects, also publishing more than 100 papers in journals, national and international conferences. His research work is focused on the communication and coordination in ubiquitous systems and context-aware applications.

María Visitación Hurtado-Torres is an Associate Professor at the University of Granada (Spain) where she received both an MSc in Computer Science and a PhD in Computing. Main fields of research: Information and Communication Technologies Applied to Health, Inclusion and Education, Assistive Technologies, Ontology Engineering and Blockchain. Around 100 research papers published in international journals and specialized conferences on these topics. Coordinator of the Master in Management and Business Process Technologies. Experience in more than 30 international, national and regional R&D&I competitively-financed projects. Editorial Board member of the Journal of Information & Management (Elsevier), and reviewer of several prestigious international conferences.