


# A Blockchain-Enabled Framework for Controlled Access to Cluster Resources


**Kausthav Pratim Kalita**

(Tezpur University, Tezpur, Assam, India)

 <https://orcid.org/0000-0001-9738-7640>, [koztov.project@gmail.com](mailto:koztov.project@gmail.com))


**Debojit Boro**

(Tezpur University, Tezpur, Assam, India)

 <https://orcid.org/0000-0001-5512-0913>, [deb0001@tezu.ernet.in](mailto:deb0001@tezu.ernet.in))

**Dhruba Kumar Bhattacharyya**

(Tezpur University, Tezpur, Assam, India)

 <https://orcid.org/0000-0002-9506-7640>, [dkb@tezu.ernet.in](mailto:dkb@tezu.ernet.in))

**Abstract: Purpose:** Big data applications enable organizations to derive actionable insights that inform strategic decision making and enhance operational efficiency in real time. Hadoop's architecture features a distributed file system that stores voluminous data across multiple machines within a cluster. However, the management and control of access to this data can be viewed as centralized, as Hadoop relies on a central coordination system to manage tasks and resources across the cluster.

**Design / methodology / approach:** To address the limitation in Hadoop, this paper proposes integrating blockchain technology to establish strict authentication procedures through smart contracts, enabling controlled access to the Hadoop platform. The proposed platform allows organizations to access Hadoop clusters through participation in a blockchain network, enabling efficient data storage mechanisms and model training capabilities.

**Findings:** The performance of this integrated system is evaluated through simulations leveraging Ethereum based smart contracts. The findings suggest that implementing appropriate indexing mechanisms and hashing techniques can enable sufficient access control, thereby facilitating controlled access to Hadoop clusters. The paper presents the simulation results in terms of execution cost and execution time.

**Originality/value:** This paper addresses the identified need for a transparent and reliable access control system that leverages blockchain's smart contracts to enable controlled and restricted access to Hadoop clusters.

**Keywords:** Access Control, Hadoop cluster, Blockchain, Smart Contract

**Categories:** C.2.4, E.2, H.1.2, H.2.4, H.2.7, H.2.8, H.3.2, H.3.3

**DOI:** 10.3897/jucs.141277

## 1 Introduction

Big data is a prominent cutting-edge technology in today's digital landscape, extending across various sectors such as business, education, finance, healthcare, and manufacturing [Chen 2020]. With its far-reaching impact on modern-day processes and operations, it encompasses a wide range of applications and has the potential to revolutionize how

organizations operate in the digital era. Its extensive influence can be observed in diverse areas, including business analytics [Sun et al. 2024], adaptive energy management [Taherdoost 2024], quality assurance monitoring in education systems [Sorour and Atkins 2024], algorithmic trading in finance [Walker et al. 2022], decision support mechanisms in healthcare [Karanastasis et al. 2024], and sustainable manufacturing [Kunecová et al. 2024]. This technology has fundamentally transformed the way data is collected, processed, and utilized, offering unprecedented insights and opportunities for innovation in the modern world. Hadoop applications play a crucial role in the field of big data by providing a scalable, cost-effective solution for storing and processing large volumes of data [Ma et al. 2023]. With the ability to handle both structured and unstructured data, Hadoop clusters are essential for organizations looking to build and manage data warehouses efficiently. The significance of Hadoop clusters extends across various industries, including finance, healthcare, retail, and telecommunications. In the finance industry, Hadoop clusters enable organizations to analyze large sets of financial data to identify trends and patterns, leading to better-informed decision-making [Shuxiang 2023]. Similarly, in healthcare, Hadoop clusters support the storage and analysis of vast amounts of patient data, contributing to improved patient care and medical research [Harb et al. 2020]. Moreover, in the retail sector, Hadoop clusters help businesses analyze customer behavior, manage inventory, and optimize pricing strategies [Verma et al. 2020]. In telecommunications, these clusters aid in processing enormous volumes of data generated by network devices and customer interactions, leading to enhanced network performance and customer experience [Kastouni and Lahcen 2022]. Ultimately, the importance of Hadoop clusters lies in their ability to handle the scale and complexity of modern data warehousing needs, providing organizations across diverse industries with the means to extract valuable insights and drive innovation.

The Hadoop Distributed File System (HDFS) serves as the primary storage system for Hadoop clusters [Yeh and Chen 2021]. It is designed to store and manage large volumes of data reliably and efficiently across a distributed network of machines [Lee et al. 2014]. HDFS provides high throughput access to application data and is suitable for workloads that have large data sets. Its significance in Hadoop lies in its ability to handle the volume, velocity, and variety of big data [Sundarakumar et al. 2021], and to provide fault tolerance and high availability. HDFS also enables data replication to ensure data durability and reliability. The architecture of HDFS allows for easy scalability, enabling organizations to expand their data storage capacity seamlessly as their data volumes grow. By dividing large files into smaller blocks and distributing them across multiple machines, HDFS ensures efficient data processing and analysis. This distributed storage model enhances both data availability and reliability, making it a critical component of the Hadoop ecosystem. HDFS enables file access using the command line or API via SSH by providing a set of command-line tools and APIs that interact with the HDFS filesystem. Users can perform various file operations such as creating, deleting, and modifying files using these tools and APIs. Using SSH for file access to HDFS adds a layer of security by allowing users to securely access HDFS from remote machines [Shrivastava and Patel 2023]. This enables users to manage and manipulate files in the HDFS cluster while ensuring secure communication and data transfer.

While Hadoop-clusters offer many advantages in terms of storage and processing capabilities, the utilization of this distributed file storage generally remains restricted to a single organization that limits the volume, variety and value of the residing data. In domains where extensive data analysis relies on larger datasets, this presents suboptimal conditions. Moreover, there is a need to explore the possibilities of enabling access to these clusters for entities that do not have the means to construct their own Hadoop

infrastructure but require a cluster environment for storing, exploring, and analyzing data for their business needs. This limitation can be addressed by integrating blockchain technology with Hadoop and HDFS to create a secure and controlled access framework for cluster resources.

While recent studies have proposed intelligent offloading and network slicing mechanisms for optimizing resource allocation in edge computing and communication network environments [Mohajer et al. 2024, Zhou et al. 2024, Yang et al. 2025], they primarily focus on performance optimization without incorporating secure and decentralized access control. Our blockchain-enabled framework ensures transparent resource management while complementing a reliable access allocation technique, thereby addressing both security and efficiency challenges in cluster-based resource allocation. Blockchain technology can play a significant role in assuring accountability and non-repudiation while managing access control, thereby creating a trustworthy environment [Wang et al. 2022]. The decentralized and immutable nature of blockchain ensures that all transactions are recorded in an irreversible sequence and cannot be altered or repudiated [Upadhyay et al. 2021]. This provides a transparent and auditable trail of access control activities, which helps in verifying the authenticity of access requests and actions taken [Kamboj et al. 2021]. In a cluster-based environment, blockchain's consensus mechanism and smart contracts can be utilized to automate and enforce access control policies, ensuring that only authorized users can access cluster resources. This scheme can not only enhance security but also instill trust in the access control process. By implementing blockchain in access control for cluster resources, organizations can create a reliable and accountable platform where the integrity of access control decisions is maintained, and non-repudiation of access activities is assured. By leveraging distributed ledger technology, this paper introduces a collaborative model utilizing the Ethereum blockchain to establish a network of peers interested in joining a Hadoop-cluster to access the infrastructure's resources. Such a participation benefits both the cluster owner and peers by creating a reliable and secure platform for storing data from various sources, thereby enhancing data analytics capabilities.

### 1.1 Organization

The rest of the paper is organized as follows: Section 2 discusses research works that focus on designing platforms where blockchain technology is integrated with big data applications. Section 3 explains the proposed methodology and its underlying components. The evaluation of the platform with respect to smart contract execution and model training is presented in Section 4, followed by concluding remarks in Section 5.

## 2 Related Works

The integration of blockchain and big data applications can offer mutually beneficial solutions to their respective challenges. Blockchain's decentralized nature addresses key problems faced by big data applications, providing solutions for ensuring data integrity and offering access control capabilities within a platform. Smart contracts can be used to design authentication measures for secured access to clusters. All actions taken by the permitted participants can also be recorded in the blockchain, maintaining a traceable record for audits. On the other hand, big data applications can address the scalability issue faced by blockchain technology by offering off-chain solutions for storing the associated data, which is indexed in the distributed ledger. Additionally, big data analytics can be

leveraged to identify patterns and anomalies in blockchain data, improving the overall performance and efficiency of blockchain networks. While exploring research on the integration of blockchain and big data applications to develop various solutions, we next discuss the closely related papers studied.

In the system proposed in [Linoy et al. 2019], Hadoop is integrated with Ethereum to address the challenges posed by the growing volume of transaction data stored on blockchain networks. Hadoop's distributed computing capabilities are leveraged to enhance the performance and scalability of querying blockchain data, enabling auditors to efficiently conduct more complex analyses and monitor for compliance and irregularities in a timely manner while preserving privacy. The paper [Chen et al. 2019] propose a blockchain-based personnel data management system that leverages the immutability and traceability of blockchain to address security concerns with centralized databases. The system features a prototype with capabilities for querying, adding, modifying, and tracking employee information, demonstrating the viability of blockchain for personnel data management. Smart contracts execute operations on the blockchain, which securely stores critical core data. A separate database stores less sensitive non-core data, with hash values stored on the blockchain to enable verification and maintain tamper-proof off-chain storage. The authors in [Demirbaga and Aujla 2022] propose a scalable computing system to manage and analyze healthcare big data from IoT devices. Their architecture includes a big data analytics tracking system and a blockchain-based data storage/access system. To handle the data volume, they suggest storing only essential data on the blockchain and verifying data integrity by comparing on-chain and off-chain data. This approach effectively addresses scale and privacy while maintaining security and verification. Their system leverages big data analytics to process, analyze, and gain insights from structured and unstructured healthcare data, supporting decision-making and predictions. The paper [Zhou et al. 2023] introduces a blockchain-based approach for securely storing and managing spatiotemporal big data collected from ordinary sensor nodes. It employs a dual strategy that enables both encrypted and non-encrypted data operations based on data types, facilitating secure data sharing through smart contracts. The framework integrates elliptic curve cryptography for efficient and secure encryption of sensitive information and leverages the InterPlanetary File System (IPFS) to enhance data storage reliability and accessibility.

The work presented in [Li et al. 2020] introduces a blockchain-based approach to verify data integrity in cloud storage without relying on a third-party auditor. This method uses blockchain to securely store lightweight verification tags, which are then employed to construct a Merkle Hash Tree for effective and reliable data integrity verification. This solution addresses the impracticality and trust issues associated with traditional integrity verification methods that require downloading entire datasets or involve cloud service providers. The authors in [Yang et al. 2020] propose a blockchain-based framework to enable secure and reliable data sharing in big data networks. The authors introduce a novel Proof of Contribution consensus mechanism and transaction filtering strategies tailored for resource-limited devices, which are crucial in scenarios where big data is collected from numerous edge devices. The paper's focus on big data is in the context of securely and efficiently managing and sharing data in edge computing environments, particularly addressing the challenges of large data volumes and the need for computational efficiency in resource-constrained settings. The paper [Amjad et al. 2022] present a blockchain-based approach to address data exchange challenges in power systems with growing renewable energy integration. The proposed architecture combines Hyperledger Fabric and Hadoop to enable secure and scalable data sharing among power system entities, addressing concerns related to third-party data handling. The research

Ref	Area	Purpose of Big Data	Big Data Application	Purpose of Blockchain	Blockchain Application	Discussion on Smart Contract
[Chen et al. 2019]	Information Management	immense volumes of personal information are managed	any traditional centralized database	core data and hashes of non-core data are stored in the blockchain	Hyperledger Fabric	chaincodes (smart contracts) are discussed with results
[Linoy et al. 2019]	Blockchain Query Processing	transaction-based blockchain data are stored for MapReduce-based query execution	Hadoop	any transactional data stored in the blocks of a traditional blockchain	Ethereum	not provided
[Li et al. 2020]	Cloud Storage	large volumes of files, documents, databases, or any other type of data are stored in the cloud	the paper discusses the general concept of using cloud storage for big data	lightweight verification tags are stored on the blockchain	a prototype system is developed	blockchain-based public auditing scheme is discussed
[Yang et al. 2020]	Big Data Transaction Platform	distributed storage of industry big data	not mentioned	access address of the data is written in the blockchain	a prototype system is developed	not provided
[Zhang et al. 2020]	Data Security Protection	the study uses metadata from namenode for role-based access management using risk value	Hadoop	the blockchain maintains the role-based access of the users using smart contracts	not mentioned	the functionality of smart contracts is discussed
[Mothukuri et al. 2021]	Provenance Traceability	large datasets are stored	Hadoop	file metadata is stored in the blockchain to verify the authenticity and track the lineage of files shared within Hadoop	Hyperledger Fabric	not provided
[Amjad et al. 2022]	Data Exchange in Power System	large amount of data generated from power systems are exchanged	Hadoop	transactions and data exchange are conducted using the blockchain	not mentioned	initial verifications conducted by the smart contract is discussed
[Demirbaga and Auja 2022]	Healthcare Management in IoT	large amount of data generated by IoT devices in healthcare sector	Hadoop	to complement the tracking system, blockchain is used to ensure that the data is securely stored and accessed.	Ethereum	data accessibility using the smart contract is discussed
[Gupta and Dwivedi 2023]	Security in Big Data Models	voluminous data generated from multiple sources	Hadoop	storage of keys for authentication purposes	not mentioned	not provided
[Zhou et al. 2023]	Management of Spatiotemporal Big Data	secured storage of spatiotemporal big data	HDFS is compared with the proposed Classified Secure Storage Technology	IPFS-based storage information is stored in the blockchain	not mentioned	not provided
[Dadkhah et al. 2024]	Management of Big Data Storage	storage of large files with minimized retrieval speed and latency	DBNode system is designed for storing large files	access control mechanism and storage-based information	Hyperledger Fabric	smart contract functionality is discussed
[Alshammari 2024]	Security of Big Data Models	big data generated from a wide range of sources are considered in their study	big data analytic platforms in general	storage of sensitive information accessible through authorization	includes Ethereum and Hyperledger Fabric in discussion	the significance of smart contracts is highlighted
Our work	Healthcare Management as a use case	data gathered from multiple machines that have been registered through a smart contract	Hadoop	blockchain manages client information and allow access to HDFS data based on permission-based allocation	Ethereum	smart contract is discussed with pseudocodes

Table 1: Related research in the field of big data and blockchains: A comparison.

work [Dadkhah et al. 2024] presents a decentralized storage system that uses erasure coding to divide large files into chunks, thereby enhancing the management of big data storage. The system also implements a two-layer hash-slot mechanism and a mirror strategy to optimize data retrieval and redundancy, ensuring high performance while meeting the unique access control needs of consortium blockchains. The protection of the content residing in the blockchain is achieved through rule-based accessibility implemented using chaincodes. The paper [Alshammari 2024] presents a thorough analysis of the security challenges in big data models and proposes a secure big data framework that leverages blockchain technology for encrypted data storage and access control. The model amalgamates traditional and modern security approaches, offering a comprehensive strategy to safeguard sensitive information. The study underscores the distinct advantages of blockchain in achieving tamper-resistance and enhanced data integrity.

The paper [Zhang et al. 2020] proposes a security framework for Hadoop by incorporating blockchain technology. The authors discuss the shortcomings in Hadoop's existing security, such as reliance on single Kerberos authentication and vulnerabilities due to a single NameNode. Their contribution includes the development of a scheme that uses a distributed cluster of NameNode servers managed by blockchain to enhance security and decentralize metadata management. The approach improves the heartbeat model for timely detection of DataNode failures and uses smart contracts to dynamically allocate user roles based on risk values and user behavior, achieving more effective access control and monitoring within the Hadoop ecosystem. The research work in [Mothukuri et al. 2021] suggests a way to improve the security and traceability of the HDFS through the incorporation of blockchain technology. The authors present BlockHDFS, which utilizes the Hyperledger Fabric platform to store file metadata from HDFS on a blockchain. This integration offers an unchangeable and permanent record of file modifications while ensuring the trustworthy origin of data. By using blockchain to track file metadata, the system ensures that file modifications are transparent and verifiable, which is crucial for investigating security breaches or unauthorized changes. The paper [Gupta and Dwivedi 2023] presents a blockchain-based three-tier authentication framework for improving security in big data applications within the Hadoop ecosystem. The framework incorporates multiple authentication methods to strengthen security. The authors highlight the use of blockchain as a tamper-proof storage solution, replacing the local database at the Key Distribution Center. This enhances data integrity and traceability, improving system security against various attacks, including replay and insider threats.

Table 1 presents a comparison highlighting the key differences in approaches taken by the related papers and our paper. The analysis suggests that big data frameworks like Hadoop can facilitate the scalability of blockchain systems [Linoy et al. 2019]. Conversely, blockchain has emerged as a preferred secure repository for storing sensitive information, owing to its tamper-resistant properties [Chen et al. 2019, Demirbaga and Aujla 2022, Zhou et al. 2023]. This characteristic has enabled the application of distributed ledger technology across diverse domains [Alshammari 2024, Gupta and Dwivedi 2023], including cloud-based platforms [Li et al. 2020], edge computing networks [Yang et al. 2020], and power systems [Amjad et al. 2022]. Furthermore, the utilization of smart contracts empowers blockchains to provide essential features such as accessibility [Dadkhah et al. 2024], authorization [Zhang et al. 2020], and traceability [Mothukuri et al. 2021]. While these research works provide valuable comprehensive solutions in their respective domains, they do not explore the possibility of facilitating a Hadoop-cluster to potential users by managing a secure access to the cluster resources. Identifying this knowledge gap, we suggest a collaborative framework called Block-

Hadoop for organizations to contribute in the expansion of data in a Hadoop-cluster, which could enhance the knowledge base and utilize machine learning and predictive modeling. With this platform, organizations can continue to extract knowledge from their data without being limited by storage or processing constraints. The key contributions of this paper are enumerated below.

1. A model, termed BlockHadoop, is designed to allow owners of Hadoop-clusters to make their big data infrastructure available to organizations that generate large volumes of data over time and require a reliable warehouse for storage purposes.
2. The blockchain technology is used to leverage the functionality of smart contracts in order to track users associated with the Hadoop-cluster. The indexing mechanism allows participants to store and access data with specific metadata more efficiently.
3. The prototype model is tested in a simulation environment by utilizing the Ethereum blockchain and Solidity-based smart contracts, along with a Hadoop-cluster. Experiments involve running the smart contract to obtain execution details, as well as applying machine learning algorithms using Scikit-Learn and Spark ML libraries to obtain performance results. The results have been found highly satisfactory.

### 3 BlockHadoop: The Proposed Framework

Big data is generated through the collection and analysis of large volumes of information from diverse origins, including the healthcare sector. Healthcare 4.0 has evolved as the next generation paradigms for healthcare, leveraging advanced technologies to transform the industry. It focuses on the integration of digital technologies, the Internet of Things, and data-driven approaches to improve patient care, operational efficiency, and decision-making processes. It emphasizes the use of big data analytics, artificial intelligence, and machine learning to enable predictive and personalized medicine, optimize resource allocation, and enhance patient outcomes. As the growing nature of medical information makes it a significant contributor to big data, the healthcare scenario has been selected to demonstrate the usefulness of our BlockHadoop framework. Medical data in a hospital is generated through various sources such as electronic health records, medical imaging systems, laboratory information systems, and wearable health devices. Interaction of patients with healthcare professionals results in the creation of data, including medical history, diagnoses, treatment plans, medications, and test results. Additionally, administrative data such as billing and insurance information also contribute to the overall medical dataset. This data can be compiled and uploaded in batches to ensure efficient and organized management. Hadoop can handle not only structured data like electronic health records and administrative data, but also unstructured data like medical images and sensor data generated by wearable devices. Our framework takes into account structured text data along with associated metadata that can later be used to train models for predictions and other data analysis-based activities.

The system model is illustrated in Fig. 1. The overall workflow process demonstrates the involvement of components and participants in the collaborative system. The Hadoop-cluster administrator (Admin), who owns the infrastructure, enables the sharing of medical data among peers (clients) through a smart contract that operates within a blockchain network. The following steps, labeled in the figure, are discussed in sequence.

1. Admin deploys a smart contract on the blockchain network to indicate the availability of the Hadoop platform for data storage.

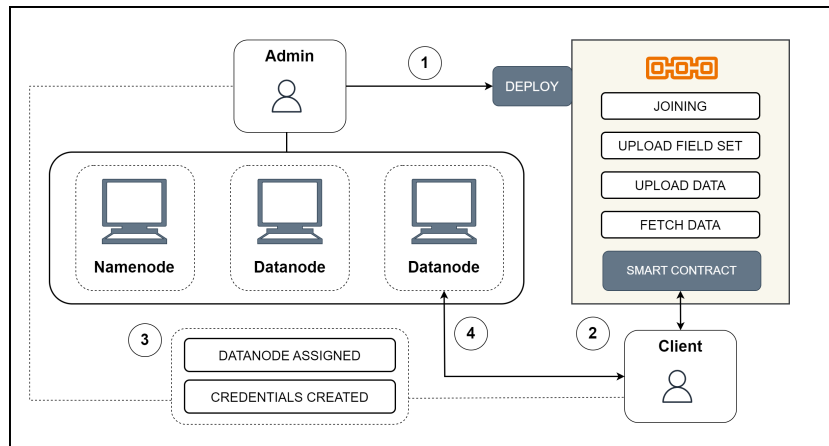


Figure 1: Conceptual framework of BlockHadoop.

2. The interested client registers to the Hadoop-cluster through a smart contract that defines the provisions offered from the platform.
3. The Admin approves the client after verification and allocates one of the connected client machines with the necessary configurations and permissions. This allows the client to perform read and write operations in the cluster.
4. The client can access the Hadoop-cluster using credentials generated by the Admin.

The platform is designed to allow any party with adequate big data storage capabilities to form a collaborative community of users by deploying a smart contract on the blockchain network. The deployer acts as both the owner of the contract and the supervisor of the Hadoop-cluster. Clients who choose to be part of the Hadoop-cluster can request through the smart contract for membership approval. Joining the network provides access to the data repository, allowing storage and retrieval of data. The Admin also benefits from contributions by various clients, as it increases the volume of the repository for the Hadoop-cluster. With a vast amount of data, Hadoop can efficiently distribute the processing workload across its nodes, leading to faster query execution and improved insights. Additionally, large volumes of data can potentially enable better predictive analytics and machine learning models, ultimately leading to better decision-making and strategic insights for the organization. Thus, the framework creates an environment that is advantageous for both the Admin and the associated clients. Once a client user sends a request for participation, the Admin configures the necessary settings for HDFS interaction and informs the client through the smart contract. After the configuration is completed, the client can access the HDFS programmatically and through command-line utilities such as *hdfs dfs*.

The Admin oversees the management of both the NameNode and DataNodes in the Hadoop-cluster. The clients are able to either upload new data or retrieve information according to specified formats or metadata. The data upload process involves initial operations such as indexing record instances and hashing batch-wise data. The data processing steps are depicted in Fig. 2. The procedure shown in the diagram is explained below.

- Step 1: For each record in a batch, the individual field values are altered by adding a certain amount of noise (gaussian noise). Once the random noise is added to all the record instance, a hashing algorithm (SHA256) is applied on the batch to obtain a unique hash value.
- Step 2: In the next stage, the data with added noise is indexed in order and the patient identity details (such as unique identifier) are removed. These indices are then compiled together with the HDFS location and the hash calculated in the previous stage.
- Step 3: Finally, the data with added noise is uploaded to the Hadoop-cluster, and the compiled data including indices, the HDFS location, and the hash value are uploaded to the blockchain via our proposed smart contract.

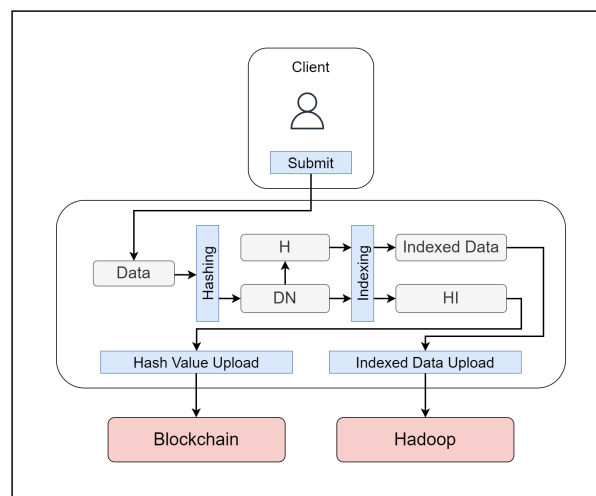


Figure 2: Data processing steps [H: Hash Value; DN: Data with Noise; HI: Hash with Indices].

The procedure is demonstrated with an example in Fig. 3. The example uses a unique identifier as the personal detail of a patient, which is removed during the indexing stage. It is crucial to remove patient unique identification numbers before sharing data in order to protect patient privacy and confidentiality. These numbers can be used to identify individuals and, if shared, may lead to unauthorized access or use of sensitive healthcare information. By removing patient identification details, the risk of privacy breaches and identity theft is significantly reduced, allowing for safer and more secure data sharing practices [Chenthara et al. 2020]. The figure illustrates that the original batch of data is kept on the client's local machine with corresponding indices tagged against each record instance for future verification if necessary.

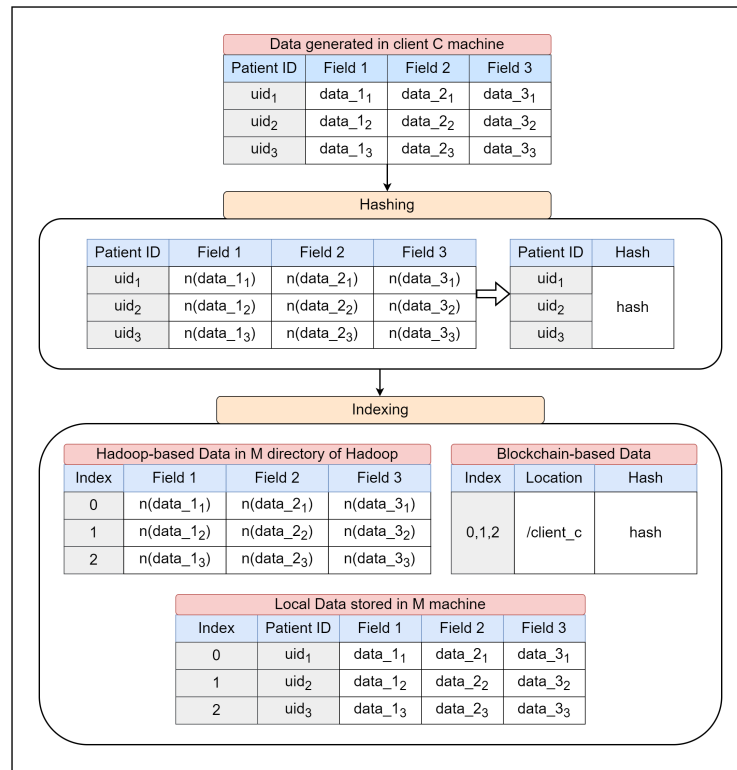


Figure 3: Demonstration of data processing with an example.

### 3.1 Authorization Management

The Admin needs to ensure that only legitimate users are granted access to the cluster, preventing unauthorized access and potential security breaches. Being the owner of the Hadoop-cluster, the Admin determines the level of access and actions that an authenticated user or client is permitted within the cluster. Thus, the authorization mechanism must define what resources and data the clients can interact with and what operations they can perform. This is crucial for maintaining data privacy, compliance with regulations, and preventing unauthorized modifications or data breaches. Our model has authorization at two levels: host and cluster. At the host level, authorization involves defining entry permissions for client users to use the client machines. These client machines can either be the DataNodes in the cluster or machines that can directly access the cluster. Authorization includes specifying which users are allowed to connect to specific hosts and for how long they can perform on those hosts. By setting up host-level authorization, organizations can enforce strict controls over the interactions and activities taking place at the host machine, adding an additional layer of security to the cluster. On the cluster level, authorization encompasses setting up policies and rules that govern access to the overall cluster resources and data. This involves defining roles and access levels for different users or groups, specifying which parts of the cluster they can interact with, and what actions they are allowed to perform. By implementing cluster-level authorization, the

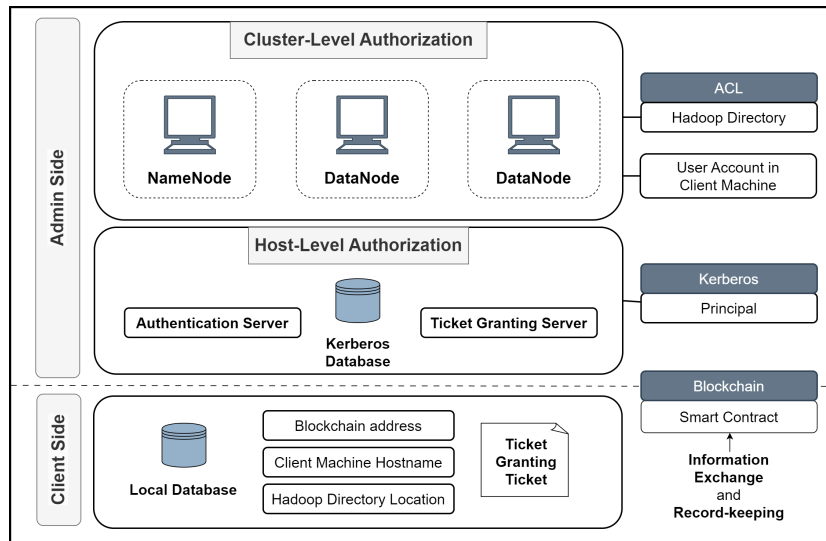


Figure 4: Authorization Management in BlockHadoop.

Admin can ensure that users have the necessary permissions to perform their tasks while preventing unauthorized activities that could compromise the integrity of the entire cluster. Fig. 4 illustrates the authorization management in BlockHadoop.

### 3.1.1 Kerberos: Host-Level Authorization

Kerberos is an authentication protocol used to verify the identity of entities accessing network resources, primarily in large networks for Single Sign-On support [Albaldawi and Almuttairi 2021]. It ensures mutual authentication between clients and servers, preventing impersonation and man-in-the-middle attacks. Kerberos is widely employed in systems requiring reliable auditing and authentication features and serves as an alternative authentication system to SSH while providing benefits such as Single Sign-On.

When a new client gives a joining request to the Admin through the designated smart contract, the Admin configures the Kerberos server to include the new client by creating a *principal* for the client. This involves generating a new key for the client and adding it to the Kerberos database using the *kadmin* utility. While configuring, the Admin defines the access policies and permissions for the new client within the Kerberos server. Here, the *principal*<sup>1</sup> represents a unique identity that can be authenticated within the system and is identified by a *principal* name, which is usually in the format of `<host>/<client_hostname>@<server_domain_name>`.

Next, the client configures its local machine to use the Kerberos infrastructure for authorization. This includes installing the necessary Kerberos client libraries and configuring its machine to communicate with the Kerberos server. During connection, the Key Distribution Center (KDC) provides the client with the necessary *keytab* file containing the client's key. This *keytab* file allows the client to authenticate with the Kerberos server

<sup>1</sup> [https://web.mit.edu/kerberos/krb5-1.5/krb5-1.5.4/doc/krb5-user/What-is-a-Kerberos-Principal\\_003f.html](https://web.mit.edu/kerberos/krb5-1.5/krb5-1.5.4/doc/krb5-user/What-is-a-Kerberos-Principal_003f.html)

and obtain tickets for accessing the client machine that can access the Hadoop-cluster. A *keytab* file is a secure repository for storing the long-term keys of principals (users or services) in the Kerberos server. The *keytab*<sup>2</sup> file allows the clients to authenticate to the Kerberos server without entering a password, making it a valuable asset in proving access to a client.

### 3.1.2 Access-control List: Cluster-Level Authorization

For a robust access control in the cluster level, the Admin uses the Access-control List (ACL) mechanism available in Hadoop, which facilitates access regulation to resources in the Hadoop ecosystem<sup>3</sup>. The Admin can use it to determine who can access specific files or directories and their permissible actions. ACLs offer more precise control over access permissions compared to traditional Unix-style permissions. With ACLs, administrators are able to assign specific permissions to users or groups, allowing for a more detailed access control approach. Their usage enables effective management of access policies within Hadoop, thereby safeguarding sensitive data and restricting modifications only to authorized users.

After the admin completes the Kerberos settings, a new user account for the client is created in a client machine. Next, a new directory is created for the client in HDFS with read and write permissions. These access permissions for the directory are granted using the *setfacl* utility in Hadoop. Once the arrangements are completed, the Admin submits the necessary details to the client through an *approval* function availed in the smart contract.

## 3.2 Smart Contract Development

Smart contracts are used to automate and enforce the terms of an agreement using blockchain technology. They provide a secure and transparent way to execute transactions without the need for intermediaries, saving time and reducing costs. Our system model's blockchain network can utilize the smart contract feature to record data uploads and retrievals on the platform. The smart contract is deployed by the Admin, and multiple other Admins can also maintain their respective smart contracts representing their Hadoop clusters. A *constructor* function includes the basic details for recognizing a specific

<sup>2</sup> [https://web.mit.edu/kerberos/krb5-1.12/doc/basic/keytab\\_def.html](https://web.mit.edu/kerberos/krb5-1.12/doc/basic/keytab_def.html)

<sup>3</sup> <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsPermissions-Guide.html>

---

#### Pseudocode 1 Constructor

---

```

parent_directory ← directory inside which the client-based directories are created
ticket_grant_time ← validity time of tickets generated using the Kerberos server
server_hostname ← hostname of the client machine that can access the cluster
procedure CONSTRUCTOR(folder_name, time, name)
    Owner = msg.sender
    parent_directory = directory_name
    ticket_grant_time = time
    server_hostname = name
procedure CLUSTER_INFORMATION()                                ▷ function to fetch cluster details
return parent_directory, ticket_grant_time, server_hostname

```

---

**Pseudocode 2** Joining request from a client

---

```

client ← struct data-type to store the client details
client_hostname ← the hostname of the client's local machine
client_map ← mapping table to store client details against client addresses
msg.sender ← address of the caller
procedure JOIN(client_hostname)
  create new client instance
  client.id = msg.sender
  client.host_name = client_hostname
  client_map[msg.sender] = client

```

---

**Pseudocode 3** Approval by the Admin

---

```

client ← struct data-type to store the client details
directory ← the new directory created in the HDFS for the requested client
ticket_grantable ← boolean value that indicates if a client is approved
start_timestamp ← approval begins at this timestamp
end_timestamp ← approval ends at this timestamp
msg.sender ← address of the caller
procedure APPROVAL(address, name)
  require(Owner == msg.sender)
  client_map[address].directory = name
  client_map[address].ticket_grantable = true
  client_map[address].start_timestamp = block.timestamp
  client_map[address].end_timestamp = block.timestamp + 30 days

```

---

**Pseudocode 4** Adding a new *header\_code*


---

```

client_list ← mapping table to store list of client information
header_code_struct ← struct data-type to store header-code and corresponding attributes
header_code_list ← list of header-code related information
msg.sender ← address of the caller
procedure ADD_HEADER_CODE(header_code, attribute_set)
  require(client_map[msg.sender].length>0)
  create new header_code_struct instance
  header_code_struct.header_code = header_code
  header_code_struct.attribute_set = attribute_set
  header_code_list.push(header_code_struct)
  emit HeaderCode_Event(header_code, attribute_set);

```

---

Hadoop-cluster. The structure of the *constructor* is shown in Pseudocode 1. Once the smart contract is deployed, interested client machines can join the blockchain network to access the HDFS. At first, the client submits a request using *join* function shown in Pseudocode 2. The Admin can configure the settings accordingly and submit approval using the *approval* function shown in Pseudocode 3. During approval, these clients are provided with the required configuration details for accessing the distributed file system.

Once membership is acquired, client users can upload data in batches to the HDFS. To facilitate collaborative work and perform machine learning based analysis, each uploaded dataset is assigned a *header-code*, which represents a unique set of attributes. When submitting batch-data, it is mandatory to reference an existing *header-code* from the distributed ledger. The pseudocode for entering a new *header-code* is provided in Pseudocode 4, while Pseudocode 5 outlines the process of submitting the data to the blockchain. Later on, the client can utilize Pseudocode 6 to fetch the list of submitted

**Pseudocode 5** Adding a new batch-record

---

```

client_list ← mapping table to store list of client information
record_struct ← struct data-type to store records
record_list ← list of record related information
msg.sender ← address of the caller
procedure ADD_BATCH_RECORD(header_code, indices, location, hash_value)
  require(client_map[msg.sender].length>0)
  for i → 0 to field_code_list.length - 1 do
    if hash(header_code_list[i].header_code) == hash(header_code) then
      create new record_struct instance
      record_struct.indices = indices
      record_struct.location = location
      record_struct.hash_value = hash_value
      record_struct.header_code = header_code
      record_list.push(record_struct)
    emit Record_Event(header_code, indices, location, hash_value);

```

---

**Pseudocode 6** Retrieve the list of *header\_codes*


---

```

header_code_list ← list of header-code related information
msg.sender ← address of the caller
procedure FETCH_HEADER_CODES()
  require(client_map[msg.sender].length>0)
  return header_code_list

```

---

**Pseudocode 7** Retrieve the list of records

---

```

record_list ← list of record related information
msg.sender ← address of the caller
procedure FETCH_BATCH_RECORDS()
  require(client_list[msg.sender].approved)
  return record_list

```

---

*header\_codes* and Pseudocode 7 to download the recorded data from the blockchain. After retrieving the records from the blockchain via the smart contract, the client can extract the desired data from the HDFS location and verify its integrity by calculating hash values of the records and comparing them with those downloaded from the blockchain.

In addition to data extraction, the platform can also be used for storing trained models after completing any machine learning tasks on the downloaded data. These stored models can then be saved back in the Hadoop-cluster along with detailed information about the parameters and datasets used during their training process. Storing machine learning models in Hadoop is beneficial for the HDFS clients because it enables them to access the models from the same platform where the data is stored. This proximity simplifies the process of deploying the models into production, as they can be readily accessed and utilized alongside the data they were trained on. Additionally, keeping the models within the Hadoop-cluster allows for efficient utilization of storage resources and facilitates seamless collaboration among the participating clients.

Alongside storing the trained model within the designated Hadoop folder, a copy is concurrently submitted to the IPFS. The storage platform returns back a hash value, which the client can submit to the smart contract through a dedicated function. Multiple details need to be uploaded alongside the model to ensure a comprehensive transfer of information. The inputs to the argument of the function must include the document name

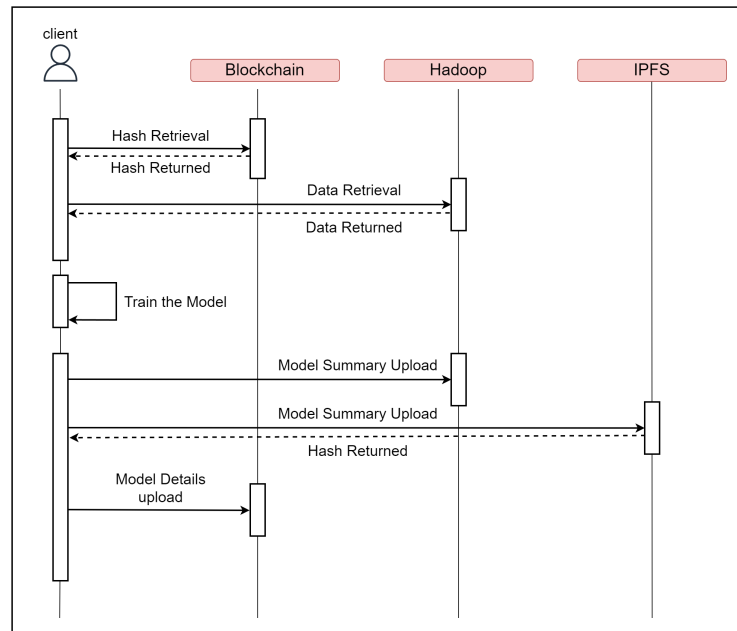


Figure 5: Sequence diagram.

to identify the file, hash value of that file for verification purpose, the IPFS-based hash value for content retrieval, the header code to obtain the list of attributes of the dataset, and the client-hash multi-valued data for obtaining the record instances. The client-hash value is the combination of the name of the client and the hashes of the batch-wise uploaded data. Additionally, the function offers the capability to include a reference to a previous block, enabling model versioning functionality. The entire procedure discussed here is demonstrated using a sequence diagram in Fig. 5. The pseudocode of this function is provided in Pseudocode 8.

---

**Pseudocode 8** Adding a new model-record
 

---

```

record_list ← list of record related information
msg.sender ← address of the caller
procedure ADD_MODEL_RECORDS(m_name, m_hash, ipfs_hash, code, data, prev_block)
  require(client_list[msg.sender].approved)
  create new model_struct instance
  model_struct.name = m_name
  model_struct.hash = m_hash
  model_struct.ihash = ipfs_hash
  model_struct.header_code = code
  model_struct.dataset = data
  model_struct.prev_block = prev_block
  model_list.push(model_struct)
  emit Model_Event(m_name, m_hash, ipfs_hash, code, data, prev_block)
  
```

---

## 4 Experiments and Results

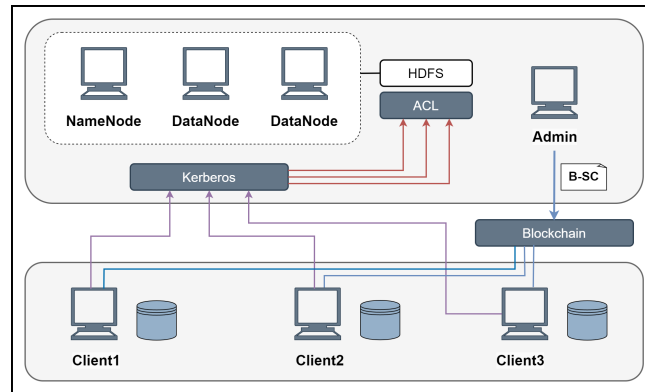


Figure 6: Simulation Testbed.

To test the performance of the proposed model, a test-bed is deployed for simulation. The setup designed for applying the strategies for BlockHadoop is provided in Fig. 6. A blockchain network is established using the Ethereum-based applications, such as Truffle and Ganache, consisting of three client machines. Truffle<sup>4</sup> and Ganache<sup>5</sup> are commonly used in blockchain simulation works for creating a local network with multiple accounts. Truffle is a development environment, testing framework, and asset pipeline for Ethereum, aiming to ease the development process of establishing a blockchain network. It provides a suite of tools that are essential for working with smart contracts and decentralized applications. On the other hand, Ganache is a personal blockchain for Ethereum development that can be used to deploy contracts, develop applications, and run tests. Together, they provide a realistic and efficient way to simulate and test blockchain networks and applications before deploying them in a live environment. The smart contract is written and tested in the Remix Ethereum IDE<sup>6</sup>, which helps developers obtain the estimated costs in deploying the contract and running the included functions. The smart contracts are written in the Solidity programming language, which is a statically-typed language that is designed for developing smart contracts that run on the Ethereum Virtual Machine. Next, the Hadoop-cluster is set up including three client machines that maintain the HDFS. A NameNode and two DataNodes are setup under the administration of the Admin user, who maintain the access control to the file system using Hadoop-based ACLs. As discussed earlier, the ACLs help to specify access permissions for the HDFS files and directories. For each client machine, two folders are created: *data* and *models*. The *data* folder is available for storing batch-wise data and the *models* folder stores any trained models submitted by the client.

<sup>4</sup> <https://archive.trufflesuite.com/docs/>

<sup>5</sup> <https://archive.trufflesuite.com/ganache/>

<sup>6</sup> <https://remix.ethereum.org/>

#### 4.1 Participation

Initiating the experiment, *BlockHadoop*'s smart contract (referred to as *B-SC*) is deployed by the Admin. The client machines join the network by providing their user details. Once the approval is attained from the Admin, the clients can participate in the blockchain network as well as use the Hadoop-cluster. The execution cost for the deployment and registration stages is depicted in Fig 7. The workflow is elaborated in the following sequential steps.

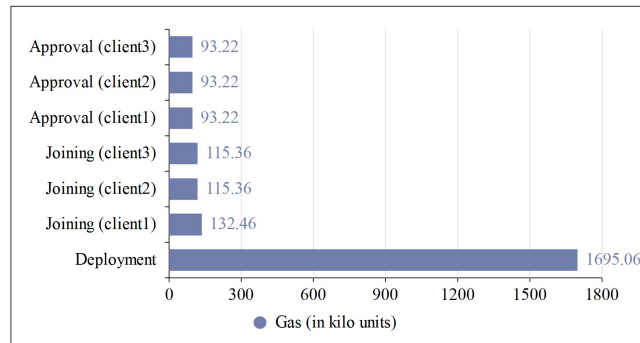


Figure 7: Gas consumption during *B-SC* deployment, joining and approval activities.

Step 1: The Admin deploys the *B-SC* in the blockchain network. This allows all the peer nodes in the network view the following details using *cluster\_information* function in Pseudocode 1.

```
parent_directory: /collab_users/
ticket_grant_time: 30 days
server_hostname: admin.bh.com
```

Step 2: The clients can view this information. They request the Admin for membership by triggering the *join* function in Pseudocode 2 with the required *client\_hostname* parameter.

```
client_hostname for 1st client: client1.bh.com
client_hostname for 2nd client: client2.bh.com
client_hostname for 3rd client: client3.bh.com
```

Step 3: The Admin creates three dedicated principles for the respective clients in the Kerberos system using *addprinc* command, which are managed by the associated KDC. The principals are as follows:

```
client1's principal:
host/client1.bh.com@BH.COM
```

```

client2's principal:
host/client2.bh.com@BH.COM
client3's principal:
host/client3.bh.com@BH.COM

```

Step 4: The Admin creates dedicated directories inside `/collab_user` (parent\_directory) for the respective clients. The ACL-based permission levels of these directories are assigned by utilizing the `setfacl` command. The folders for the three clients are shown below.

```

client1's directory: /collab_users/client1
client2's directory: /collab_users/client2
client3's directory: /collab_users/client3

```

These folders contain two sub-folders: *data* and *model*. Finally, the Admin triggers the *approval* function (Pseudocode 3) in B-SC.

Step 5: After the approval, the clients become collaborators of the Hadoop-cluster. When a client wants to authenticate itself to the Kerberos system, it requests a ticket from the KDC. The KDC then verifies the user's identity and issues a ticket-granting-ticket (TGT) if the authentication is successful. The TGT can then be utilized to request service tickets for specific services within the realm, including SSH. This allows users to enter into the client machines with authorization using SSH without needing to provide any credentials.

## 4.2 Data Uploading Process

The Pima Indians Diabetes dataset is utilized for testing the pseudocodes outlined in Section 3 and for recording execution time during machine learning tasks. The dataset has 768 records with nine features: *Pregnancies*, *Glucose*, *BloodPressure*, *SkinThickness*, *Insulin*, *BMI*, *DiabetesPedigreeFunction*, *Age*, and *Outcome*. A new dummy feature, referred as *PatientID*, is added to the list. However, it is removed in the data pre-processing procedure. This dataset is split into three parts (*pima1*, *pima2* and *pima3*), assuming they are generated by the three clients separately. Since the list of attributes remain same for the three clients, these nine features are collectively represented by one single *header\_code*. The detailed information of the three clients are shown in Table 2.

At first, the *header\_code* is submitted by one of the clients using the *add\_header\_code* function as presented in Pseudocode 4. This entry is visible to the remaining clients and, thus, need not be submitted again. The recorded gas consumption for uploading the *header\_code* (with length=86) in 10 sequential rounds is shown in Fig. 8. It also shows the execution cost of fetching the uploaded *header\_codes* after each submission. The graph indicates linear growth of execution cost with respect to the continuous entry of the given *header\_code*. However, in practical scenario, a unique *header\_code* needs to be uploaded only once. Thus, the gas consumption will vary based on the length of the attribute list. Each client then processes their individual datasets as batch-data through the hashing step and the noise addition. While testing, the data is applied with gaussian noise, as it is often a good choice because it reflects the natural variability and uncertainty present in real-world data [Momeny et al. 2021]. In the next step, the

client <sup>1</sup>	dataset	header_code	attributes	instances	indices	location	hash <sup>3</sup>
client1	pima1				1-256	/collab_users/client1	hash1
client2	pima2	code1	nine attributes <sup>2</sup>	256 each	257-512	/collab_users/client2	hash2
client3	pima3				513-768	/collab_users/client3	hash3

<sup>1</sup> client1's address: 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB  
 client2's address: 0x583031D1113aD414F02576BD6afaBfb302140225  
 client3's address: 0xD870fA1b7C4700F2BD7f44238821C26f7392148

<sup>2</sup> nine attributes: pregnancies, glucose, bpressure, skinthickness, insulin, bmi, dpfunction, age, outcome.

<sup>3</sup> hash1 = 80517ed322316bb48e9bfe14ec35ce7830383c68e6e6058c7e19da5540ce8ca  
 hash2 = 167675c45da5d5a058fb3d38d4b463dd26a63d0fed8536caacb08e7b86a212dc  
 hash3 = 90a4018e3fe728ded14958a5afb162944738768d1d0fdd6348727a186a742d7b

Table 2: Submission details of the three clients.

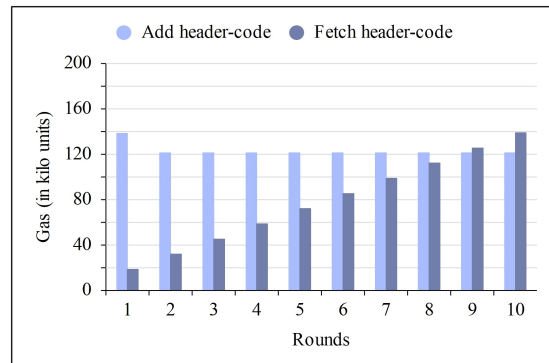


Figure 8: Execution cost of uploading and downloading header-code data.

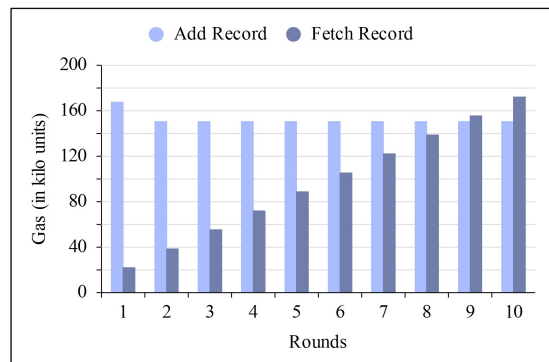


Figure 9: Execution cost of uploading and downloading batch-wise data.

processed data is indexed and the range of these indices are considered as inputs to the distributed ledger. Submitting the range instead of the index values substantially reduces the gas consumption as it decreases the length of the input parameter to B-SC. The Hadoop-based data is then submitted by each client to its designated directory and the entered data is accessible to the other two clients. Alongside, the Ethereum-based data

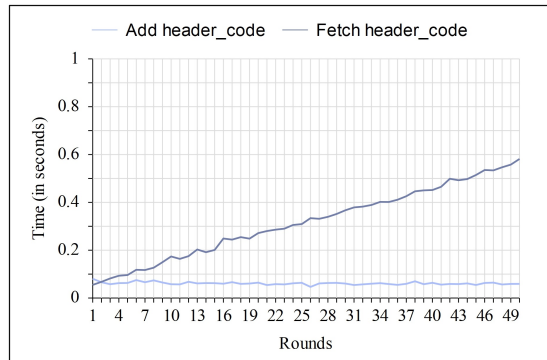


Figure 10: Execution time of uploading and downloading `header_code` data.

with `header_code` is uploaded to the blockchain using the `add_batch_record` function presented in Pseudocode 5. Each entry is accessible to the other two clients and can be fetched using the `fetch_record` function described in Pseudocode 6. Fig. 9 showcases the gas consumption of uploading `pimal` dataset in 10 occasions and downloading them subsequently after each submission. The results exhibit the same pattern as observed in case of `header_codes` with a slight increase in gas due to the larger input sizes and the inclusion of iteration to check the presence of a `header_code` before accepting the submitted record. Next, the execution time recorded while triggering the functions in B-SC are shown in Fig. 10 and Fig. 11. The graphs obtained from the experiments show a gradual increase in time for upload and retrieval of both the `header_codes` and the hash-related data. Here, uploading and downloading the batch-wise data takes more time due to the comparatively larger volume of content.

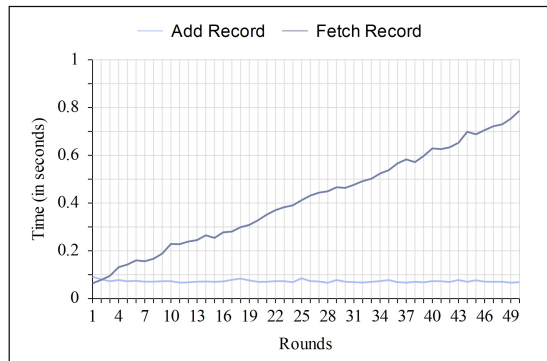


Figure 11: Execution time of uploading and downloading batch-wise data.

Accessing the blockchain's contents with specific requirements, such as header codes and batch-wise data, necessitates iterating through the relevant data structures. To facilitate efficient and resource-effective access, the necessary functions are coupled with event calls. These events enable the logging and monitoring of contract activity,

which is crucial for users to track real-time state changes. Furthermore, events promote auditing and analysis by providing an auditable record of contract behavior. In the BlockHadoop framework, events are written for logging the activities related to the submission of header codes (referred as *HeaderCode\_Event*) and batch-wise data (referred as *Record\_Event*). Both these events are called by the *emit* keyword included in Pseudocode 4 and Pseudocode 5. Two related examples of log content are shown in Fig. 12 and Fig. 13.

```
AttributeDict({'args': AttributeDict({'code': 'code1',
  'attributes': 'pregnancies, glucose, bpressure, skinthickness, insulin, bmi, dpfunction, age, outcome'})),
  'event': 'HeaderCode_Event',
  'logIndex': 0,
  'transactionIndex': 0,
  'transactionHash': HexBytes('0x9413b06fd257683dd64e4d4bf98149d7218303f51a5d2c5bea0ac9993f6fc156'),
  'address': '0xaC42A3d3317231F4A01cb7F5D2A6E7144ed6d3F6',
  'blockHash': HexBytes('0xa3085b3ec97523c32af2b5a15310590c12f700c95efacaec83d2b3d319b90868'),
  'blockNumber': 5})
```

Figure 12: HeaderCode Event log output that includes the code name, list of attributes, the event name, and the block number.

```
AttributeDict({'args': AttributeDict({'code': 'code1',
  'indices': '1-256',
  'hash': '80517ed322316bb48e9bfde14ec35ce7830383c68e6e6058c7e19da5540ce8ca'})),
  'event': 'Record_Event',
  'logIndex': 0,
  'transactionIndex': 0,
  'transactionHash': HexBytes('0x52a8c654045167e1ae4918be044cb946dc07b027c6481d14549459f0bba419e5'),
  'address': '0xaC42A3d3317231F4A01cb7F5D2A6E7144ed6d3F6',
  'blockHash': HexBytes('0x1565fd58aa397b3cd027dd930efe7f95b82610be85bb4bc5cd1f6c42335d78f9'),
  'blockNumber': 6})
```

Figure 13: Record Event log output that includes the code name, range of indices, hash value, event name, and the block number.

### 4.3 Model Training

The Admin and the three clients have the ability to access the Pima dataset<sup>7</sup> from HDFS and utilize Spark ML<sup>8</sup> libraries to acquire trained models. Spark ML's integration with HDFS allows for seamless data access and processing, making it a valuable tool for the clients in the blockchain network. The significance of Spark ML in a Hadoop-cluster lies in its ability to leverage the distributed computing power of the Hadoop ecosystem for machine learning tasks. By running Spark ML on a Hadoop-cluster, the Admin and the clients can efficiently handle large-scale data processing and build and deploy machine

<sup>7</sup> <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

<sup>8</sup> <https://spark.apache.org/docs/1.2.2/ml-guide.html#main-concepts>

learning models at scale. Through the utilization of B-SC, the *fetch\_record* function can be utilized by the clients to retrieve record instances with matching header\_codes and run popular machine learning libraries like Scikit-Learn<sup>9</sup> to train models outside HDFS. To demonstrate this utility, both Spark ML and Scikit-Learn are applied for training the dataset residing in HDFS. As the dataset is suitable for binary classification, four supervised learning algorithms are used: logistic regression, decision tree, random forest, and gradient boost. The overall time required for fetching and training the models are shown in Fig. 14.

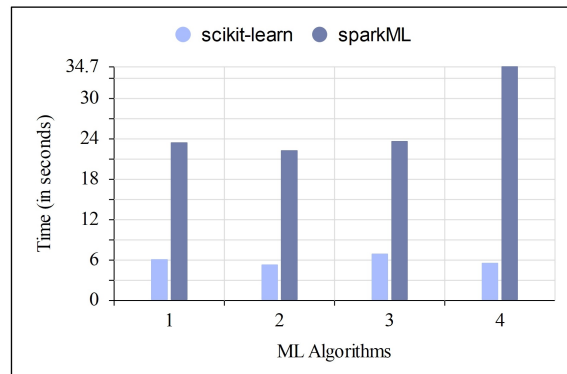


Figure 14: Execution time of training the models using Scikit-Learn and Spark ML libraries. [1: logistic regression; 2: decision trees; 3: random forest; 4: gradient boost]

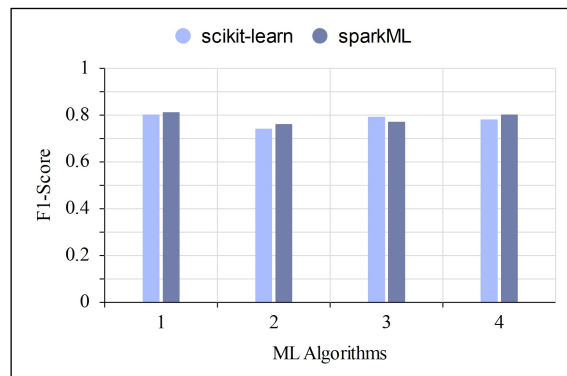


Figure 15: Comparison of the F1-Scores after training the models using Scikit-Learn and Spark ML libraries. [1: logistic regression; 2: decision trees; 3: random forest; 4: gradient boost]

<sup>9</sup> <https://scikit-learn.org/stable/>

```

AttributeDict({'args': AttributeDict({'model_name': 'log_reg.h5',
'ipfs_hash': 'QmYHgDGnUb2wP5L47NL75VN9k2UTNXwzH5fpF6HTsbMEui',
'code': 'code1',
'dataset': 'client1-hash1, client2-hash2',
'previous_block': 6}),
'event': 'Model_Event',
'logIndex': 0,
'transactionIndex': 0,
'transactionHash': HexBytes('0x51feea3c5e777a434c25473413bb47d0788ecbc779204f51e93a67bcdab3d1ca'),
'address': '0xAD6a8c9f3B9D844Ae5c565Ab3daCbA4b72d46AeA',
'blockHash': HexBytes('0x8e0b21a5546cb48fa45122bffaeb8bb425eae3ef6cf7b4ed9d5c61fc479d20a7'),
'blockNumber': 24})

```

Figure 16: Model Event log output that includes model-related details, event name, and the block number.

The reason behind faster execution of Scikit-Learn libraries is due to the fact that it is primarily a Python library and is optimized for single-machine performance, whereas Spark ML is built on top of the Apache Spark framework, which is designed for distributed computing. Additionally, Scikit-Learn uses optimized C and Cython code for its core computations [Subasi 2020], while Spark ML relies on the JVM for its operations [Omar and Jumaa 2022]. This fundamental difference in design and implementation leads to Scikit-Learn being faster for many machine learning tasks. Thus, the client can choose one of the approach based on the size of the dataset. As it is an unbalanced dataset, F1-score values are selected for comparing their performances, which is shown in Fig. 15. The output results show the similarity of performance between both the approaches.

Similar to *HeaderCode\_Event* and *Record\_Event*, a dedicated event (referred as *Model\_Event*) is attached to Pseudocode 8 that records the details related to a newly trained model or an updated model. It holds the essential information of the submission such as the IPFS hash, the data selected for training the model and reference to a previous block if a model is updated. As shown in Fig. 16, the log output from Block No. 24 demonstrates that the submitted model utilized the hash1 dataset of client1 and the hash2 dataset of client2 for the training process. Additionally, the log indicates that the current model is referring to a previous model stored in Block No. 6.

## 5 Conclusion

Hadoop has excelled in the field of big data for several years due to its scalability, flexibility, and cost-effectiveness. Its distributed computing model and fault tolerance feature enable it to handle massive datasets and complex analytics with ease as data volumes continue to grow exponentially. BlockHadoop leverages Hadoop's potential by integrating it into an ecosystem where blockchain enables parties to join a cluster and participate in the distributed repository. The proposed B-SC smart contract ensures secure access to the cluster with minimal data ingestion from users associated with it. The owner of the Hadoop-cluster holds ultimate authority over granting or denying permissions based on resource availability while interactions between owners and clients are recorded on the blockchain network, ensuring transparency and auditability of the platform. However, attention must be given to growing volume of data in the distributed ledger which can impact performance when burdened by large participation or multiple user interactions simultaneously. Additionally, this platform is limited to structured data which may not

cover use-cases involving semi-structured and unstructured data. Furthermore, exploring integration possibilities with other popular big-data and blockchain applications could offer greater versatility alongside optimal security measures.

## Disclosure statement

There is no potential conflict of interest to be disclosed.

## Funding

No funding has been received.

## References

- [Albaldawi and Almuttairi 2021] Albaldawi WS, Almuttairi RM. Kerberos authentication for big data applications on cloud environment. In *Journal of Physics: Conference Series* 2021 Feb 1 (Vol. 1804, No. 1, p. 012062). IOP Publishing.
- [Alshammari 2024] Alshammari A. Secure Big Data Model Based on Blockchain Technology. *Advances in Science and Technology. Research Journal.* 2024;18(4):163-76.
- [Amjad et al. 2022] Amjad M, Taylor G, Lai CS, Huang Z, Li M. A Novel Blockchain Based Approach to Exchanging Information and Data in Power Systems. In *2022 57th International Universities Power Engineering Conference (UPEC) 2022 Aug 30* (pp. 1-6). IEEE.
- [Chen et al. 2019] Chen J, Lv Z, Song H. Design of personnel big data management system based on blockchain. *Future generation computer systems.* 2019 Dec 1;101:1122-9.
- [Chen 2020] Chen Y. IoT, cloud, big data and AI in interdisciplinary domains. *Simulation Modelling Practice and Theory.* 2020 Jul 1;102:102070.
- [Chenthara et al. 2020] Chenthara S, Ahmed K, Wang H, Whittaker F, Chen Z. Healthchain: A novel framework on privacy preservation of electronic health records using blockchain technology. *Plos one.* 2020 Dec 9;15(12):e0243043.
- [Dadkhah et al. 2024] Dadkhah N, Ma X, Wolter K, Wunder G. DBNode: A Decentralized Storage System for Big Data Storage in Consortium Blockchains. In *2024 9th International Conference on Big Data Analytics (ICBDA) 2024 Mar 16* (pp. 270-279). IEEE.
- [Demirbaga and Aujla 2022] Demirbaga U, Aujla GS. MapChain: A blockchain-based verifiable healthcare service management in IoT-based big data ecosystem. *IEEE Transactions on Network and Service Management.* 2022 Sep 6;19(4):3896-907.
- [Gupta and Dwivedi 2023] Gupta MK, Dwivedi RK. Beaf: BD—A Blockchain Enabled Authentication Framework for Big Data. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal.* 2023 Dec 29;12:e19163-.
- [Harb et al. 2020] Harb H, Mroue H, Mansour A, Nasser A, Motta Cruz E. A hadoop-based platform for patient classification and disease diagnosis in healthcare applications. *Sensors.* 2020 Mar 30;20(7):1931.
- [Kamboj et al. 2021] Kamboj P, Khare S, Pal S. User authentication using Blockchain based smart contract in role-based access control. *Peer-to-Peer Networking and Applications.* 2021 Sep;14(5):2961-76.

- [Karanastasis et al. 2024] Karanastasis E, Andronikou V, Chondrogiannis E, Tagaris A, Mourt-zoukos K, Psychas A, Varvarigou T. Data-empowered clinical trial design and eligible patient selection through the PONTE platform. *Simulation Modelling Practice and Theory*. 2019 May 1;93:245-61.
- [Kastouni and Lahcen 2022] Kastouni MZ, Lahcen AA. Big data analytics in telecommunications: Governance, architecture and use cases. *Journal of King Saud University-Computer and Information Sciences*. 2022 Jun 1;34(6):2758-70.
- [Kunecová et al. 2024] Kunecová J, Bikfalvi A, Marques P. Sustainability orientation, industrial big data and product innovation: Evidence from the European manufacturing sector. *Computers & Industrial Engineering*. 2024 Apr 18:110163.
- [Lee et al. 2014] Lee JS, Ye SK, Jeong HD. ATMSim: An anomaly teletraffic detection measurement analysis simulator. *Simulation Modelling Practice and Theory*. 2014 Dec 1;49:98-109.
- [Li et al. 2020] Li J, Wu J, Jiang G, Srikanthan T. Blockchain-based public auditing for big data in cloud storage. *Information Processing & Management*. 2020 Nov 1;57(6):102382.
- [Linoy et al. 2019] Linoy S, Mahdikhani H, Ray S, Lu R, Stakhanova N, Ghorbani A. Scalable privacy-preserving query processing over ethereum blockchain. In 2019 IEEE International Conference on Blockchain (Blockchain) 2019 Jul 14 (pp. 398-404). IEEE.
- [Ma et al. 2023] Ma C, Zhao M, Zhao Y. An overview of Hadoop applications in transportation big data. *Journal of traffic and transportation engineering (English edition)*. 2023 Sep 30.
- [Mohajer et al. 2024] Mohajer A, Hajipour J, Leung VC. Dynamic offloading in mobile edge computing with traffic-aware network slicing and adaptive TD3 strategy. *IEEE Communications Letters*. 2024 Nov 18.
- [Momeny et al. 2021] Momeny M, Neshat AA, Hussain MA, Kia S, Marhamati M, Jahanbakhshi A, Hamarneh G. Learning-to-augment strategy using noisy and denoised data: Improving generalizability of deep CNN for the detection of COVID-19 in X-ray images. *Computers in Biology and Medicine*. 2021 Sep 1;136:104704.
- [Mothukuri et al. 2021] Mothukuri V, Cheerla SS, Parizi RM, Zhang Q, Choo KK. Block-HDFS: Blockchain-integrated Hadoop distributed file system for secure provenance traceability. *Blockchain: Research and Applications*. 2021 Dec 1;2(4):100032.
- [Omar and Jumaa 2022] Omar HK, Jumaa AK. Distributed big data analysis using spark parallel data processing. *Bulletin of Electrical Engineering and Informatics*. 2022 Jun 1;11(3):1505-15.
- [Shrivastava and Patel 2023] Shrivastava G, Patel S. Secure Storage and Data Sharing Scheme Using Private Blockchain-Based HDFS Data Storage for Cloud Computing. *International Journal of Computer Networks and Applications*. 2023 Jan 1:28-38.
- [Shuxiang 2023] Shuxiang Z. Application of Hadoop cloud platform based on soft computing in financial accounting budget control. *Soft Computing*. 2023 Jun 26:1-2.
- [Sorour and Atkins 2024] Sorour A, Atkins AS. Big data challenge for monitoring quality in higher education institutions using business intelligence dashboards. *Journal of Electronic Science and Technology*. 2024 Jan 3:100233.
- [Subasi 2020] Subasi A. Practical machine learning for data analysis using python. Academic Press; 2020 Jun 5.
- [Sun et al. 2024] Sun P, Yuan C, Li X, Di J. Big data analytics, firm risk and corporate policies: Evidence from China. *Research in International Business and Finance*. 2024 Apr 18:102371.
- [Sundarakumar et al. 2021] Sundarakumar MR, Mahadevan G, Somula R, Sennan S, Rawal BS. An Approach in Big Data Analytics to Improve the Velocity of Unstructured Data Using MapReduce. *International Journal of System Dynamics Applications (IJSDA)*. 2021 Oct 1;10(4):1-25.
- [Taherdoost 2024] Taherdoost H. A Systematic Review of Big Data Innovations in Smart Grids. *Results in Engineering*. 2024 Apr 21:102132.

- [Upadhyay et al. 2021] Upadhyay A, Mukhuty S, Kumar V, Kazancoglu Y. Blockchain technology and the circular economy: Implications for sustainability and social responsibility. *Journal of cleaner production*. 2021 Apr 15;293:126130.
- [Verma et al. 2020] Verma N, Malhotra D, Singh J. Big data analytics for retail industry using MapReduce-Apriori framework. *Journal of Management Analytics*. 2020 Jul 2;7(3):424-42.
- [Walker et al. 2022] Walker T, Davis F, Schwartz T. Big Data in Finance: An Overview. *Big Data in Finance: Opportunities and Challenges of Financial Digitalization*. 2022 Oct 4:3-9.
- [Wang et al. 2022] Wang S, Hu Y, Qi G. Blockchain and deep learning based trust management for internet of vehicles. *Simulation Modelling Practice and Theory*. 2022 Nov 1;120:102627.
- [Yang et al. 2020] Yang J, Wen J, Jiang B, Wang H. Blockchain-based sharing and tamper-proof framework of big data networking. *IEEE Network*. 2020 Jul 22;34(4):62-7.
- [Yang et al. 2025] Yang J, Mohajer A. Multi objective constellation optimization and dynamic link utilization for sustainable information delivery using PD-NOMA deep reinforcement learning. *Wireless Networks*. 2025 Feb;31(2):1839-59.
- [Yeh and Chen 2021] Yeh T, Chen Y. Improving the hybrid cloud performance through disk activity-aware data access. *Simulation Modelling Practice and Theory*. 2021 May 1;109:102296.
- [Zhang et al. 2020] Zhang C, Li Y, Sun W, Guan S. Blockchain based big data security protection scheme. In 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC) 2020 Jun 12 (pp. 574-578). IEEE.
- [Zhou et al. 2023] Zhou B, Zhao J, Chen G, Yin Y. Research on Secure Storage Technology of Spatiotemporal Big Data Based on Blockchain. *Applied Sciences*. 2023 Jul 6;13(13):7911.
- [Zhou et al. 2024] Zhou G, Mohajer A. Blind reconfigurable intelligent surfaces for dynamic offloading in fixed-NOMA mobile edge networks. *International Journal of Sensor Networks*. 2024;46(3):142-60.