

A
Project Report
on
**UNVEILING UNAUTHORIZED ACCESS IN HADOOP DISTRIBUTED
FILE SYSTEMS**

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

By

**Myana Amitha
(20EG105331)**

**Nandini Vemuganti
(20EG105333)**

**Paida Samhitha
(20EG105336)**



Under the guidance of

Dr. J BALARAJU M.Tech., Ph.D
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Venkatapur (V), Ghatkesar (M), Medchal (D), T.S - 500088

2023-24



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled “**Unveiling Unauthorized Access in Hadoop Distributed File Systems**” being submitted by **Myana Amitha** bearing the Hall Ticket number **20EG105331**, **Nandini Vemuganti** bearing the Hall Ticket number **20EG105333**, **Paida Samhitha** bearing the Hall Ticket number **20EG105336** in partial fulfilment of the requirements for the award of the degree of the **Bachelor of Technology in Computer Science and Engineering in Anurag University** is a record of bonafide work carried out by them under my guidance and supervision from academic year 2023 to 2024.

The results presented in this project have been verified and found to be satisfactory. The results embodied in this project report have not been submitted to any other University for the award of any other degree or diploma.

Internal Guide

Dr. J Balaraju
M.Tech., Ph.D
Assistant Professor

Dean, CSE

Dr. G Vishnu Murthy

External Examiner

DECLARATION

We hereby declare that the project work entitled “**Unveiling Unauthorized Access in Hadoop Distributed File Systems**” submitted to the **Anurag University** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology (B. Tech)** in Computer Science and Engineering is a record of an original work done by us under the guidance of **Dr. J. Balaraju, Assistant Professor** and this project work have not been submitted to any other university for the award of any other degree or diploma.

Myana Amitha
20EG105331

Nandini Vemuganti
20EG105333

Paida Samhitha
20EG105336

Place: Anurag University, Hyderabad

Date:

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Dr. J. Balaraju** , Assistant Professor, Department of Computer Science and Engineering, Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for his encouragement and timely support in our B. Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Department of Computer Science and Engineering, Anurag University.

We also express our deep sense of gratitude to **Dr. V. V. S. S. S. Balaram**, Academic coordinator. **Dr. T. Shyam Prasad**, Assistant Professor, Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

Myana Amitha
20EG105331

Nandini Vemuganti
20EG105333

Paida Samhitha
20EG105336

ABSTRACT

In the rapidly evolving digital landscape, managing large datasets securely and efficiently has become a critical challenge. This project aims to address the challenge by developing a system that leverages Hadoop's Distributed File System (HDFS) with Role-Based Access Control (RBAC) for secure and efficient file operations. This system is crucial for scenarios where data security and user access management are paramount. The crux of this project lies in balancing high levels of security with user accessibility in HDFS. Existing systems often struggle with this balance, either sacrificing security for accessibility or vice versa. This solution integrates a robust RBAC mechanism to control user access without complicating the user experience, ensuring that authorized users can access data efficiently and securely. This methodology involves setting up a Hadoop environment, configuring it for RBAC, and integrating these components in a Java environment for added security. Secure Shell (SSH) has for a secure remote access and operation. This comprehensive system aims to enhance the security and efficiency of file operations in HDFS. The "Hadoop RBAC Manager" outlines a scenario focused on managing RBAC within a Hadoop environment. In distributed computing frameworks like Hadoop, securing data and managing user permissions are critical challenges. The RBAC model simplifies the administration of security policies by assigning permissions to roles rather than individual users, ensuring that only authorized users can access specific data sets or perform certain operations. Overall, the project aims to improve the security and efficiency of managing large datasets in HDFS by implementing a balanced approach to security and user accessibility through RBAC.

Keywords: HDFS, Role Based Access Control(RBAC), Secure Shell(SSH)

TABLE OF CONTENTS

S. No.	CONTENT	PAGE NO
1.	Introduction	01
	1.1 Background and Importance of System	01
	1.2 Object & Scope of the Project	01
	1.3 Problem Statement	03
2.	Literature Review and Background Study	04
	2.1 Overview of Hadoop Ecosystem and Its Importance	04
	2.2 Existing Technologies	06
	2.2.1 Processing Techniques	
	2.2.2 Use of machine learning	
	2.3 Limitations in Existing systems	07
	2.3.1 Accuracy and Real-Time Processing Issues	08
	2.3.2 User Interface and Accessibility Concerns	
	2.4 Security Innovations in Hadoop Clusters	09
3.	Proposed Method	12
	3.1 Apache Hadoop and HDFS Overview	13
	3.2 RBAC (Role Based Access Control)	14
	3.3 Managing Data in HDFS	14
	3.3.1 Initial Setup and Configurations	15
	3.3.2 Uploading Files to HDFS	16
	3.3.3 Deleting Files from HDFS	16
	3.3.4 Downloading Files from HDFS	16
4.	Implementation	
	4.1 Introduction to System Requirements	17
	4.2 System Environment	18

4.3 Functional Requirements	19
4.3.1 Use Case Descriptions	19
4.3.2 List of Functional Requirements	19
4.4 Non-Functional Requirements	19
4.5 Hardware & Software Requirements	23
4.6 Data Preprocessing Techniques	25
4.7 Model Training and Optimization	28
4.8 Implementation using Helper Functions	30
5. Experimental Results	34
5.1 Initial Setup and Configuration	34
5.2 Hadoop Installation	35
5.3 Helper Function Outputs	36
6. Discussion of Results	40
6.1 Key Findings and Outcomes	40
6.2 Comparison with Existing System	42
7. Summary, Conclusion and Recommendation	43
8. Future Enhancements	44
9. References	45

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
2.1.1	Key components of RBAC	6
2.1.2	Flowchart depicting RBAC	6
4.6.1	Diagram representing working of RBAC	27
4.6.2	Working of RBAC	27
5.1.1	Installation and Setting up Java	34
5.1.2	SSH Configuration	34
5.2.1	Install and Unpack Hadoop	35
5.3.1	Before uploading files	36
5.3.2	User uploading file	36
5.3.3	User uploading file	36
5.3.4	Uploading local file to hdfs	36
5.3.5	Local file uploaded	36
5.3.6	List of files inside hdfs after uploading	37
5.3.7	Intruder do not have permission to delete the file	37
5.3.8	Owner grants permission to the user to delete the file	37
5.3.9	Original user reading the data	37
5.3.10	Original data	38
5.3.11	Intruder trying to read the data	38
5.3.12	False data	38
5.3.13	Granting permission to the other user to read the files	39

5.3.14	Original data	39
5.3.15	Logged file	39
6.1.1	Comparison of Execution Time	40
6.1.2	Comparison of File Deletion Security	41
6.1.3	Comparison of Handling Unauthorized Access	41

LIST OF ABBREVIATIONS

ABBREVIATIONS	FULLFORM
RBAC	Role Based Access Control
HDFS	Hadoop Distributed File System
SSH	Secure Shell
API	Application Interface
YARN	Yet Another Resource Negotiator
HIPPA	Health Insurance Portability and Accountability Act
GDPA	General Data Protection Regulation

1. Introduction

1.1 Background and Importance

Hadoop is an open-source framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. However, managing access to data and resources in such an environment can be complex due to the scale and distributed nature of Hadoop.

Role-Based Access Control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. RBAC helps in securing data within a Hadoop cluster by ensuring that only authorized users can access specific data sets or perform certain operations.

1.2 Objective & Scope of the Project

Objective:

To develop a system that facilitates the management of access permissions for files stored on HDFS, leveraging the principles of Role-Based Access Control (RBAC). The system aims to ensure that only authorized users can perform specific actions (e.g., reading, writing, deleting) on the files, based on their assigned roles or permissions. This is intended to enhance security, compliance, and data management efficiency within Hadoop environments.

Scope:

1. Hadoop Setup and Configuration:

Installation and configuration of Hadoop, SSH (for secure communication), and Java dependencies necessary for Hadoop's operation. This includes setting up environmental variables, Hadoop's core and HDFS configurations, and ensuring that Java is correctly configured for Hadoop's use.

2. Secure SSH Configuration:

Configuration of SSH to facilitate secure communication without the need for manual intervention during authentication. This includes generating SSH keys, configuring SSH to bypass strict host checking, and setting up authorized keys for seamless access.

3. Hadoop File System (HDFS) Initialization and Management:

Initializing the HDFS name node and data nodes, formatting HDFS, and starting HDFS daemons. The operations include managing the HDFS directory structure and permissions, enabling basic file operations like uploading, deleting, and downloading files.

4. RBAC Implementation for File Access on HDFS:

Implementing rudimentary RBAC functionalities to manage access to files stored on HDFS. This involves creating functions to upload, delete, and download files with checks to ensure that operations are performed only by users with the appropriate permissions. The system uses environment variables to manage user roles and permissions, ensuring that file operations respect the security constraints defined by the RBAC model.

5. User Interaction and Automation:

Providing a user-friendly interface and automated processes for interacting with the HDFS and managing file access. This includes for setting up the environment, functions that abstract complex operations, and integration with the Google Colab interface for file uploads and downloads.

1.2 Problem Statement

The most significant challenge within Hadoop's Distributed File System (HDFS) striking the right balance between security and user accessibility. Often, existing systems face a dilemma, having to choose between prioritizing stringent security measures or ensuring ease of use for users. Heightened security can sometimes result in a less user-friendly system, while prioritizing user-friendliness may inadvertently introduce vulnerabilities into data security. Our solution aims to overcome this challenge by implementing a robust access control mechanism based on Role-Based Access Control (RBAC). This approach allows us to maintain a delicate equilibrium, ensuring data security while keeping the system accessible and user-friendly. By integrating RBAC, we can grant appropriate permissions to users based on their roles, thus safeguarding data integrity without complicating the user experience. This balance is crucial for maintaining the integrity of vast datasets and ensuring that authorized users can access them efficiently and effortlessly.

The crux of our project lies in addressing a dual challenge in Hadoop's Distributed File System (HDFS): achieving high levels of security without sacrificing user accessibility and ease of use. Existing systems often find themselves in a trade-off between these two crucial aspects. On the one hand, enhancing security measures can lead to a system that is less accessible and user-friendly. On the other, focusing solely on user-friendliness might lead to vulnerabilities in data security. Our solution seeks to establish a balance by integrating a robust access control mechanism through RBAC without complicating the user experience. This balance is vital in maintaining the integrity of large data sets and ensuring that they are accessible to authorized users in an efficient and user-friendly manner.

2. Literature Survey

It includes steps for installing Java, configuring SSH for Hadoop's secure communication, installing Hadoop, setting up the Hadoop Distributed File System (HDFS), and creating an environment for file management in HDFS. This setup is intended for educational or development purposes, allowing users to experiment with Hadoop and its file system without the need for a dedicated cluster.

2.1 Overview of Hadoop Ecosystem and Its Importance

Hadoop Ecosystem Overview

Apache Hadoop is an open-source framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The core of Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster.

Role-Based Access Control (RBAC)

RBAC is a policy-neutral access-control mechanism defined around roles and privileges. It reduces the complexity and cost of security administration in large-scale systems. Roles are assigned based on the responsibilities and qualifications of users within an organization and the permissions to perform certain operations are assigned to specific roles.

Hadoop and Security

Initially, Hadoop was not designed with a strong emphasis on security. As it began to be adopted by large organizations with sensitive data, the need for enhanced security mechanisms became apparent. Hadoop's security features have evolved, introducing mechanisms for authentication, authorization, accounting, and data protection. However, managing access control in a granular and efficient manner remains a challenge, especially in environments where data sensitivity and user roles are dynamic.

Hadoop RBAC Manager

The Hadoop RBAC Manager concept revolves around the idea of managing access to resources in Hadoop ecosystems through roles. This involves defining roles according to the data access needs of different users or groups within an organization and assigning permissions to these roles rather than individual users. A robust RBAC system for Hadoop would need to interface with Hadoop's security components, such as Kerberos for authentication and Apache Ranger or Apache Sentry for authorization.

Challenges:

Scalability: As clusters grow, managing roles and permissions must remain efficient and not become a bottleneck.

Complexity: The diversity of Hadoop ecosystem components (HDFS, YARN, Hive, HBase, etc.) adds complexity to access control management.

Dynamic Environment: The need to quickly adjust roles and permissions in response to changing organizational structures or security policies.

Importance of Hadoop RBAC Manager

Role-based access control (RBAC) is a critical component of an organization's security strategy, enhancing data protection by allowing only authorized users to access specific data sets. By defining roles and associating them with permissions, RBAC ensures that individuals have the appropriate level of access based on their job responsibilities. This approach not only improves security but also helps organizations meet regulatory requirements, such as those outlined in GDPR or HIPAA, by controlling access to sensitive information.

Furthermore, RBAC simplifies the management of user permissions by centralizing control, reducing the administrative burden of manually assigning permissions to individual users. This centralized approach also minimizes the risk of errors, ensuring that users have the access they need to perform their jobs effectively without compromising security. Overall, RBAC enhances security, ensures compliance with regulations, and improves operational efficiency by streamlining access management processes.

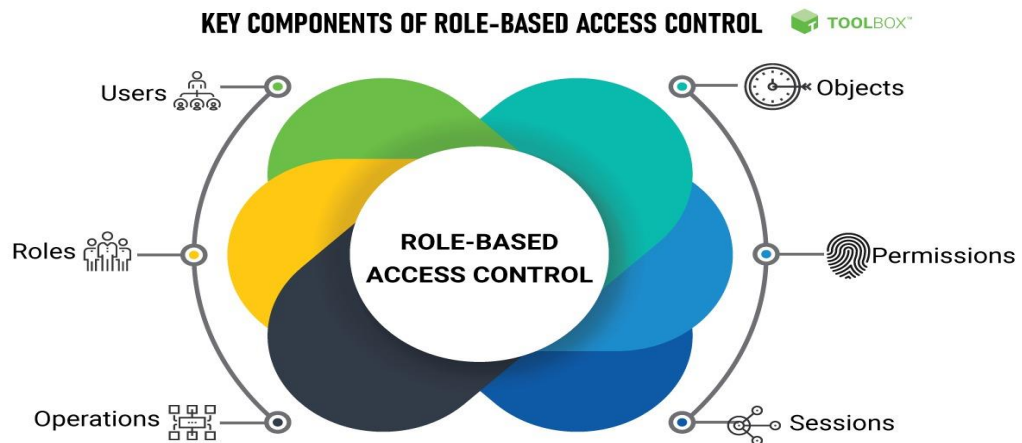


Fig 2.1.1 Key Components of Hadoop



Fig 2.1.2 Depicting RBAC

2.2 Existing Technologies

Existing Technologies on Hadoop RBAC Manager

In a sophisticated Hadoop RBAC Manager, operations are part of a comprehensive system that includes detailed role definitions, policy management, and audit and compliance features. Specific roles, such as administrator, data scientist, and data engineer, are defined with corresponding sets of permissions. Policies are established to dictate the actions each role can or cannot perform within the Hadoop Distributed File System (HDFS), including access to directories and specific operations.

Additionally, the system includes robust audit and compliance functionalities to track user actions and changes to roles or permissions. This tracking is essential for security auditing and ensuring compliance with regulatory requirements. To enhance security and RBAC capabilities, tools and extensions like Apache Ranger and Apache Sentry are commonly used with Hadoop. These tools provide fine-grained access control, advanced policy management, and auditing capabilities that are seamlessly integrated with Hadoop's ecosystem, further enhancing security and compliance measures.

Role-Based Access Control (RBAC) in Hadoop is a fundamental security framework that regulates access to resources based on the roles assigned to users. This approach is essential for efficiently managing permissions across a potentially large number of users and resources within a Hadoop cluster. The RBAC model simplifies permission management in several key ways. First, it involves defining roles and associating specific data access permissions and actions (such as read, write, and execute) with these roles rather than with individual users. Second, users are assigned roles that automatically grant them the associated permissions, making it easier to manage permissions by modifying roles rather than individual user permissions. Lastly, RBAC facilitates auditing and compliance efforts by allowing for easier tracking of data access and manipulation through role assignments and permissions, which is crucial for ensuring compliance with data protection regulations.

2.3 Limitations in Existing Systems

2.3.1 Accuracy and Real-Time Processing Issues

Hadoop RBAC Manager Role-Based Access Control (RBAC) in Hadoop is a security mechanism that manages access to resources based on the roles assigned to users. This model ensures that only authorized users can perform certain actions within the HDFS ecosystem, enhancing the security and governance of the data stored in HDFS.

Accuracy in Hadoop RBAC Manager

Accuracy in the context of a Hadoop RBAC Manager is crucial for ensuring the correct definition, assignment, and enforcement of user roles and permissions. Issues related to accuracy can stem from various factors, including misconfiguration of roles, propagation delays, and inconsistent states within the distributed environment. Misconfiguration of roles, such as defining incorrect permissions or roles, can result in unauthorized access or denial of access to legitimate users. For instance, if a user is mistakenly assigned more permissions than intended, they may gain access to sensitive data. Moreover, propagation delays can occur when changes in roles or permissions are not immediately reflected across the entire Hadoop ecosystem, particularly in large, distributed environments. This delay can lead to temporary access issues, where users might experience disruptions or inconsistencies in their access permissions. Additionally, ensuring consistency across all nodes in a distributed environment after updates to roles or permissions can be challenging. Inconsistent states can arise, leading to accuracy issues where the system's state does not align with the intended access controls.

Real-Time Processing Issues

Real-time processing in Hadoop, which involves analyzing and processing data as it arrives, is crucial for time-sensitive applications. However, RBAC management can impact real-time processing in several ways. One significant concern is the performance overhead introduced by RBAC checks. Implementing RBAC requires additional computation to verify user permissions every time data is accessed, potentially slowing down the processing time and increasing latency. Moreover, scalability challenges can arise as the number of users and roles grows. Managing and enforcing RBAC policies for a large number of users and roles can become increasingly complex and resource-intensive, potentially affecting the system's ability to scale effectively and maintain real-time processing speeds. Additionally, real-time applications often require dynamic access control, where permissions need to be adjusted on-the-fly based on the context. Implementing such dynamic RBAC policies without impacting processing speed presents a significant challenge. Overall, while RBAC is essential for ensuring secure access to data, its implementation in real-time processing environments requires careful consideration to minimize performance impacts and ensure scalability.

Mitigating Accuracy and Real-Time Processing Issues

To mitigate the impact of RBAC on real-time processing in Hadoop, several strategies can be employed. One approach is to automate the role and permission assignment process using policy definitions that can be programmatically applied. This reduces human error and ensures consistent role definitions across the system.

Another strategy is to cache permissions at various layers of the Hadoop ecosystem to reduce the performance overhead of frequent RBAC checks. It's essential to update the cache in real-time or near-real-time to maintain accuracy. Additionally, designing the RBAC system to be scalable from the beginning is crucial. This involves using scalable databases for storing roles and permissions and ensuring that the RBAC enforcement mechanism can handle high throughput with low latency.

Implementing comprehensive audit logs and monitoring systems is also vital. These tools can track access and role changes, helping to quickly identify and correct any inaccuracies in role assignments and permissions. Regular testing of the RBAC system for both performance and accuracy, including stress testing, ensures that real-time processing requirements are met even as the system scales. By implementing these strategies, organizations can minimize the impact of RBAC on real-time processing in Hadoop and maintain efficient and secure data operations.

2.4 Security Innovations in Hadoop Clusters

[1] Big Data Security Challenges: Hadoop Perspective. International Journal of Pure and Applied Mathematics

Authors: Saraladevi, B., Pazhaniraja, N., Paul, P.V., Basha, M.S.S. and Dhavachelvan, P.

The paper presents three approaches to enhance security within the Hadoop Distributed File System (HDFS):

Kerberos Mechanism: This approach utilizes the Kerberos network authentication protocol to secure connections within HDFS. It involves using tickets to authenticate nodes and clients, ensuring that only authorized users can access data blocks. The Ticket Granting Ticket (TGT) and Service Ticket play crucial roles in authenticating connections between clients and the NameNode.

Bull Eye Algorithm Approach: This novel approach focuses on ensuring the security of sensitive information stored within Hadoop. The Bull Eye Algorithm scans data nodes in a 360° view to identify any risks or vulnerabilities. By implementing this approach, it aims to provide comprehensive security coverage for data stored within HDFS, minimizing the chances of data breaches or unauthorized access.

Namenode Approach: This approach addresses the issue of data availability in the event of NameNode failures. By deploying redundant NameNodes within the HDFS cluster, it aims to ensure continuous data availability and mitigate the risk of data loss. The NameNode Security Enhance (NNSE) system manages these redundant NameNodes, allowing for seamless failover and ensuring that data remains accessible even during NameNode failures. This approach enhances the overall reliability and availability of data within HDFS, contributing to a more robust security posture.

[2] Innovative Secure Authentication Interface for Hadoop Cluster Using DNA Cryptography: A Practical Study

Authors: J. Balaraju and P. V. R. D. Prasada Rao

The paper discusses the significance of handling big data in today's distributed networks and the widespread use of Hadoop for this purpose due to its support for parallel processing and distributed computing. However, it highlights the lack of built-in data security in Hadoop clusters (HC), which poses a risk, especially for sensitive information. Existing security mechanisms rely on third-party providers, leading to computational overhead. To address this, the paper proposes a novel security mechanism called Secure Authentication Interface (SAI) layered over HC. SAI provides user authentication, metadata security, and access control, reducing computational overhead compared to existing mechanisms. The paper includes a comprehensive literature review of related work, outlining various security challenges and proposed solutions in the field. It concludes by highlighting the importance of SAI in creating a trusted environment within Hadoop clusters and suggests future research directions for securing both data and cluster management.

[3] Dynamic Node Identification Management in Hadoop Cluster Using DNA

Authors: J. Balaraju and P. V. R. D. Prasada Rao

The paper discusses Distributed Computing (DC) and its application in bioinformatics, particularly in DNA sequence analysis. It proposes a Distributed Bioinformatics Computing System to analyze DNA sequences efficiently across multiple computers. Additionally, it introduces Hadoop Cluster (HC) as a distributed processing framework and Dynamic Host Configuration Protocol (DHCP) for managing IP addresses within a network. The paper addresses the problem of dynamic node identification management in HC and presents a solution using a security layer implemented within DHCP. This security layer utilizes DNA cryptography to generate unique keys for each node, enhancing the security of the distributed system. The proposed method aims to prevent unauthorized access and ensure data security in distributed computing environments.

Author(s)	Strategies	Advantages	Disadvantages
Saraladevi, B., Pazhaniraja, N., Paul, P.V., Basha, M.S.S. and Dhavachelvan.	Kerberos Approach, Bull Eye Algorithm Approach, Name node Approach.	Secure Authentication, 360-Degree View, Redundancy for Availability.	Complexity, Resource Intensive, Resource Overhead.
Balaraju.J., PVRD Prasada Rao.	DNA Algorithm, BABA.	24x7 Security, Reduced Dependence on External Data Centers.	Performance Overhead, Complexity, Security issues.
Balaraju.J., PVRD Prasada Rao.	Secure Authentication Interface (SAI), DNA Cryptography.	Improved Performance, Reduced Overhead	Complex Implementation, Limited Compatibility.
Balaraju.J., PVRD Prasada Rao.	Integration of HDFS with Secure-Node.	Enhanced Security, Reduced Management Overhead.	Single Point of Failure, Resource Utilization.

3. Proposed Method

3.1 Apache Hadoop and HDFS Overview

Hadoop and HDFS

Apache Hadoop is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It's designed to scale up from single servers to thousands of machines, each offering local computation and storage.

HDFS (Hadoop Distributed File System) is part of the Apache Hadoop project. It's designed to store very large data sets reliably and to stream those data sets at high bandwidth to user applications.

HDFS Components

NameNode:

The NameNode is the central node in an HDFS cluster. It manages the metadata for all the files and directories stored in the cluster, including information about the location of the data blocks on the DataNodes.

DataNode:

DataNodes are responsible for storing the actual data blocks in HDFS. They manage read and write requests from clients and report back to the NameNode periodically with information about the data blocks they are storing.

Secondary NameNode: Despite its name, the Secondary NameNode is not a backup NameNode. Instead, it assists the NameNode by periodically merging the namespace and edits files. It helps in minimizing the downtime of the NameNode in case of failure.

How Data is Stored in HDFS

HDFS divides files into blocks, typically 128 MB or 256 MB in size. These blocks are then distributed across the DataNodes in the cluster.

HDFS replicates each block multiple times (default is 3) and stores these replicas on different DataNodes. This provides fault tolerance, as data can still be accessed if some DataNodes fail.

HDFS is rack-aware, meaning it tries to place replicas on different racks for fault tolerance and to reduce network traffic.

The metadata, which includes the file-to-block mapping and the location of each block's replicas, is stored in the NameNode. This allows the system to quickly locate and access the data blocks when requested by clients.

3.2 RBAC (Role-Based Access Control)

RBAC is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In the context of Hadoop, RBAC helps manage who has access to what within the Hadoop ecosystem, which is crucial for maintaining data security and governance.

RBAC is a method of restricting system access to authorized users. In the context of Hadoop, this concept can be applied to control who can read, write, or delete data in HDFS. This includes functions to upload, delete, and download files from HDFS with a focus on user permissions.

RBAC policies control access to resources based on roles assigned to users. This simplifies administration by allowing permissions to be managed in terms of job functions rather than individual user accounts.

Hadoop RBAC Manager

While Hadoop itself does not have an explicit "RBAC Manager," tools like Apache Ranger or Cloudera Sentry offer robust security management for Hadoop, including RBAC capabilities. These tools enable administrators to define access policies, enforce them across the Hadoop ecosystem, and audit access.

In such managed environments, datasets are managed with specific considerations. Data sensitivity is crucial, requiring classification of data based on sensitivity levels and applying appropriate access controls accordingly. Understanding data access patterns is also essential, as it helps tailor access controls accurately to different roles interacting with the data. Additionally, meeting audit and compliance requirements is paramount, necessitating the maintenance of detailed access logs to ensure alignment with data governance policies and regulations.

These tools and considerations are fundamental in maintaining a secure and compliant data environment within the Hadoop ecosystem, ensuring that data access is controlled, monitored, and audited appropriately.

System Operations and Management

Managing the availability of Hadoop's distributed filesystem (HDFS) involves starting and stopping its services, which can be achieved using the `start-dfs.sh` and `stop-dfs.sh` commands, respectively. These commands are crucial for system administrators to control the HDFS service status, ensuring it is available when needed and conserving resources when not in use.

In addition to starting and stopping services, another critical task is formatting the HDFS namenode. This step, performed using the `hdfs namenode -format` command, is necessary before using HDFS for the first time or after a complete reset. Formatting the namenode initializes the HDFS filesystem metadata, ensuring that it is in a consistent and usable state for storing and retrieving data. It is essential to understand that formatting the namenode is a destructive operation, as it erases all metadata and data in the HDFS, so it should be done with caution and only when absolutely necessary.

System administrators should familiarize themselves with these commands and procedures to effectively manage the availability and reliability of Hadoop's distributed file system for their organization's data storage and processing needs.

3.3 Managing Data in HDFS:

3.3.1 Initial Setup and Configurations:

1. **Java and SSH Configuration:** It begins by installing Java and configuring SSH, both essential for Hadoop's operation. Java is the programming language in which Hadoop is written, and SSH is used for secure communication between the nodes in the Hadoop cluster.

2. **Hadoop Installation and Environment Setup:** This proceeds to download and configure Hadoop, including setting environment variables such as `JAVA_HOME` and `HADOOP_HOME`, which are necessary for Hadoop to locate Java and for system commands to recognize Hadoop binaries, respectively.

3. **HDFS Configuration:** Next, it configures HDFS properties such as the default filesystem URL, replication factor, and directories for the NameNode and DataNode. This step is crucial for defining how HDFS will store data across the cluster.

4. **SSH Key Generation and Configuration:** Generating SSH keys and configuring SSH to bypass strict host checking are preparatory steps for allowing seamless communication between the master and worker nodes without manual intervention.

3.3.2 Uploading Files to HDFS:

The ``upload_to_hdfs`` function serves as a pivotal mechanism for facilitating the upload of files to the Hadoop Distributed File System (HDFS), ensuring that only authorized users can contribute data to the filesystem. This function operates by first verifying the existence of the requisite directories within the HDFS infrastructure. If these directories are not found, the function dynamically creates them, thus establishing a structured framework for organizing uploaded data. This proactive directory management process ensures the integrity and coherence of the filesystem, facilitating efficient data storage and retrieval operations. Upon confirming the presence of the necessary directories, the function proceeds to upload the specified file to the designated location within HDFS. By enforcing this stringent directory validation and creation process, the function reinforces data governance practices, mitigates the risk of data fragmentation, and fosters a standardized approach to file organization within the Hadoop ecosystem. Furthermore, by restricting file uploads to only authorized users, the function bolsters security measures, safeguarding against unauthorized data contributions and potential breaches of confidentiality. Overall, the function plays a pivotal role in promoting data integrity, security, and organizational efficiency within the Hadoop environment, offering a robust framework for controlled file uploads and structured file system management.

The function not only facilitates uploads directly to Hadoop Distributed File System (HDFS) but also supports file uploads from local sources, enhancing flexibility and accessibility. This capability streamlines the data ingestion process, enabling users to seamlessly contribute data stored on their local machines to HDFS. By accommodating uploads from local files, the function promotes collaboration and data sharing across distributed teams, consolidating disparate datasets within the centralized Hadoop environment. Furthermore, it enhances data accessibility, allowing users to contribute data irrespective of their physical location or network connectivity, which is particularly advantageous for remote personnel. Overall, this feature enriches the function's functionality, facilitating seamless data integration and empowering users to leverage their local data assets within the Hadoop ecosystem.

3.3.3 Deleting Files from HDFS:

The deletion function, ``delete_from_hdfs``, parallels the file upload process by adhering to RBAC (Role-Based Access Control) principles, reinforcing data governance and security within the Hadoop Distributed File System (HDFS). Like its upload counterpart, this function rigorously verifies the ownership of the file against the current user, validating permissions before allowing deletion. By enforcing this ownership validation step, the function safeguards against unauthorized data removal attempts, preserving data integrity and preventing potential data loss. This approach aligns with RBAC principles, which prioritize access control based on predefined roles and permissions, ensuring that only authorized users with the appropriate privileges can execute file deletion operations. Furthermore, by integrating RBAC principles into the deletion process, the function reinforces the overarching security framework of the Hadoop ecosystem, mitigating the risk of data breaches and unauthorized access. Through its adherence to RBAC principles, the ``delete_from_hdfs`` function not only facilitates controlled data removal but also promotes accountability and transparency in data management practices. Overall, this function serves as a critical component in enforcing access control policies, upholding data governance standards, and fostering a secure and compliant data environment within HDFS.

3.3.4 Downloading Files from HDFS:

The download function within the Hadoop Distributed File System (HDFS) adheres to stringent access control measures, akin to its counterparts for file upload and deletion. By validating file ownership against the current user, this function ensures that data access is granted only to authorized individuals with the requisite permissions. However, in instances where the user lacks the necessary privileges to access the file, the function employs a clever strategy. Rather than outright denying access, this method subtly presents a simulated file, to the unauthorized user. The simulated file serves as a decoy, mimicking the appearance and structure of the original file while containing innocuous or fabricated data. By leveraging plausible files, the download function effectively mitigates the risk of unauthorized access attempts without alerting the user. This approach not only preserves the confidentiality and integrity of sensitive data but also provides a valuable opportunity to detect and monitor unauthorized access attempts discreetly. Furthermore, the utilization of security mechanisms underscores the sophistication and versatility of access control mechanisms within HDFS, demonstrating a proactive approach to security that goes beyond traditional access denial techniques.

4. Implementation

4.1 Introduction to System Requirements

Introduction

The "Hadoop RBAC Manager" refers to managing access to HDFS resources based on user roles, aligning with the principle of least privilege. This involves ensuring that users can only perform actions (upload, download, delete) on HDFS resources for which they have permissions. This showcases basic RBAC through conditional access and operations based on the user's role or identity, which is crucial for securing data in a multi-user Hadoop environment.

Importance in Project Lifecycle:

In the project lifecycle, the requirement specification serves as a crucial blueprint, guiding the development process in the right direction. It outlines the project's scope, objectives, and functionalities, helping to avoid scope creep and ensuring that the project stays on track. By clearly defining what needs to be achieved, the requirement specification facilitates effective resource allocation, ensuring that resources are utilized efficiently to meet project goals.

Furthermore, the requirement specification serves as the basis for testing and validation. It establishes the criteria against which the developed system will be tested to ensure that it meets the specified requirements. This document enables the testing team to design test cases and scenarios that cover all aspects of the system, ensuring thorough testing and validation. By providing clear guidelines for testing, the requirement specification helps to verify the functionality, performance, and quality of the system, ultimately contributing to the project's success.

4.2 System Environment

Initial Setup and Configuration

1. Java and SSH Configuration:

It begins by installing Java and configuring SSH, both essential for Hadoop's operation. Java is the programming language in which Hadoop is written, and SSH is used for secure communication between the nodes in the Hadoop cluster.

2. Hadoop Installation and Environment Setup:

This proceeds to download and configure Hadoop, including setting environment variables such as **JAVA_HOME** and **HADOOP_HOME**, which are necessary for Hadoop to locate Java and for system commands to recognize Hadoop binaries, respectively.

3. HDFS Configuration:

Next, it configures HDFS properties such as the default filesystem URL, replication factor, and directories for the NameNode and DataNode. This step is crucial for defining how HDFS will store data across the cluster.

4. SSH Key Generation and Configuration:

Generating SSH keys and configuring SSH to bypass strict host checking are preparatory steps for allowing seamless communication between the master and worker nodes without manual intervention.

4.3 Functional Requirements

4.3.1 Use Case Descriptions

Use Case for Hadoop RBAC Manager

The use case for a Hadoop RBAC Manager emerges from the necessity to efficiently manage who has access to what data within a Hadoop environment. In enterprises that deal with vast amounts of sensitive or critical data, such as financial institutions, healthcare organizations, and government agencies, the need to enforce strict access controls cannot be overstated. The imperative for a Hadoop RBAC Manager stems from the imperative to efficiently administer access to data within Hadoop environments, particularly in sectors handling extensive sensitive information like finance, healthcare, and government. With a Hadoop RBAC Manager, organizations can implement granular access controls, tailoring permissions to specific roles and directories within HDFS. This ensures that users only interact with data pertinent to their organizational roles, thereby mitigating the risk of unauthorized access or data breaches. Moreover, the RBAC system facilitates seamless auditability and compliance with regulations such as GDPR and HIPAA by enabling organizations to track data access patterns. Through role-based access control, the system enhances overall security posture, reducing the likelihood of data leaks. Additionally, automation inherent in RBAC streamlines administrative tasks, optimizing operational efficiency, particularly in dynamic or large-scale environments. As an integral component of data governance frameworks, the Hadoop RBAC Manager ensures

adherence to organizational policies and standards, bolstering data integrity and regulatory compliance. In essence, by governing who can perform specific actions within HDFS, the Hadoop RBAC Manager establishes a robust foundation for secure and compliant data management in today's data-centric enterprises.

4.3.2 List of Functional Requirements

The Hadoop RBAC Manager is a conceptual system devised to meticulously administer and enforce access controls within a Hadoop ecosystem, centered on user roles. Its primary objective is to restrict users' access to data and operations pertinent to their organizational roles, thereby enhancing both security and operational efficiency. The detailed functional requirements of the Hadoop RBAC Manager encompass various aspects crucial for its effective implementation and operation. These include comprehensive role management functionalities, encompassing role definition, permission assignment, modification, and deletion. Additionally, user management capabilities ensure seamless user registration, role assignment, modification, and deactivation or deletion. Authentication and authorization mechanisms are vital, ensuring secure user verification and access control based on assigned roles and permissions. Furthermore, robust access logging and auditing functionalities are integral for monitoring and analyzing user activities, aiding in security audits and compliance checks. Integration with core Hadoop components such as HDFS and YARN is essential, enabling control over file access and job submission based on user roles. Finally, the system's high availability and scalability ensure uninterrupted operation and performance scalability in the face of increasing user counts and request volumes. Collectively, these requirements delineate the Hadoop RBAC Manager's breadth and capabilities, striving to furnish a secure, efficient, and user-centric solution for access control management within the Hadoop ecosystem.

4.4 Non-Functional Requirements

Non-functional requirements (NFRs) define the attributes of the system that are not related to specific functionalities but are crucial for the overall system performance, usability, security, and maintainability. These requirements include aspects such as reliability, scalability, security, performance, usability, and compliance.

Non-Functional Requirements (NFRs) are the criteria that judge the operation of a system, in contrast to the behaviors specified by Functional Requirements. They are often referred to as "qualities" of a system.

1. Security

Ensuring stringent access controls aligned with predefined roles stands as a fundamental imperative for the system, guaranteeing that users are restricted to actions permitted by their designated roles. Alongside access control, safeguarding sensitive data remains paramount, necessitating robust encryption and other security measures to protect data integrity both at rest and in transit. Authentication and authorization mechanisms must be robust to authenticate users securely and authorize their activities based on their assigned roles, mitigating the risk of unauthorized access. Furthermore, comprehensive auditability is essential, mandating the logging of all user actions and modifications to access controls, thus providing a detailed audit trail crucial for compliance adherence and forensic analysis. These aspects collectively establish a robust framework for access management, data protection, and accountability within the system, fostering a secure and trustworthy environment for organizational operations.

2. Performance and Scalability

Optimizing latency is imperative for the RBAC Manager, ensuring swift execution of critical functions such as authentication, authorization, and role adjustments to maintain a seamless user experience. By minimizing latency, the system can swiftly process user requests without causing delays that could impede user productivity or hinder system responsiveness. Furthermore, scalability is essential to accommodate the expanding user base and increasing data volumes without compromising performance. The RBAC Manager must seamlessly scale to support large Hadoop clusters, ensuring that as the system grows, it can efficiently handle the concurrent demands placed upon it. Additionally, resource efficiency is paramount to prevent undue strain on system resources such as CPU and memory, which could otherwise impact the overall performance of the Hadoop ecosystem. By optimizing resource usage, the RBAC Manager can operate efficiently within the Hadoop environment, ensuring that its functions seamlessly integrate with other system components while maintaining optimal performance levels. These considerations collectively contribute to a robust and responsive RBAC system capable of meeting the dynamic needs of modern Hadoop deployments.

3. Compatibility and Integration

Ensuring compatibility and seamless integration is paramount for the RBAC Manager within the Hadoop ecosystem. Compatibility mandates that the RBAC Manager harmoniously interacts with key components such as HDFS, YARN, and Hive, accommodating their unique access control requirements. By aligning with these foundational elements, the RBAC Manager can effectively govern data access and user permissions across the Hadoop infrastructure, fostering consistency and security. Integration capabilities extend beyond the Hadoop ecosystem, necessitating smooth interaction with existing enterprise systems. Integration with systems like LDAP for authentication is crucial, enabling organizations to leverage their existing infrastructure investments while ensuring a cohesive user experience. Furthermore, support for standard protocols enhances deployment flexibility, simplifying the integration process and facilitating interoperability with a wide range of systems and technologies. In essence, robust compatibility and integration capabilities enable the RBAC Manager to seamlessly integrate into existing IT environments, maximizing its effectiveness and utility across diverse organizational landscapes.

4. Reliability and Availability

Reliability and availability are paramount for the RBAC Manager, necessitating robust measures to ensure uninterrupted operation and data integrity. Fault tolerance is a cornerstone, requiring the system to withstand individual component failures without compromising overall functionality. By designing the RBAC Manager with fault tolerance in mind, organizations can maintain high availability even in the face of unforeseen disruptions, thereby minimizing downtime and ensuring consistent access control management. Additionally, comprehensive backup and recovery mechanisms are indispensable for safeguarding critical RBAC configurations and facilitating swift restoration in the event of system failures. By regularly backing up configuration data and implementing efficient recovery processes, the RBAC Manager can swiftly recover from incidents, mitigating potential data loss and ensuring business continuity. Together, these reliability and availability measures fortify the RBAC Manager, providing organizations with confidence in its ability to sustain operations and uphold access control functionalities under various circumstances.

5. Maintainability and Support

Maintainability and ongoing support are essential facets of the RBAC Manager's lifecycle, necessitating a modular design and proactive maintenance approach. Modularity is paramount, requiring the system to be structured in a modular fashion, enabling straightforward updates and maintenance activities. By embracing modularity, organizations can efficiently manage system modifications and enhancements, minimizing disruptions to ongoing operations. Furthermore, robust support and update mechanisms are vital for ensuring the RBAC Manager remains resilient against emerging security threats and evolving functional requirements. Regular updates, coupled with active support services, enable organizations to promptly address security vulnerabilities and implement functional enhancements, thereby enhancing the system's efficacy and resilience over time. Together, these maintainability and support measures empower organizations to sustain the RBAC Manager's performance and adaptability, ensuring it remains a reliable cornerstone of their access control infrastructure.

Fulfilling these NFRs is crucial for the successful deployment and operation of a Hadoop RBAC Manager within an organization's IT ecosystem. They ensure that the RBAC Manager not only performs its core functions effectively but also contributes to the overall security, efficiency, and manageability of the Hadoop platform.

4.5 Hardware & Software Requirements

Definition: Hardware and software requirements specify the necessary hardware components, software tools, libraries, and frameworks required to develop, deploy, and run the system

Hardware Requirements

CPU: Google Colab provides machines with either a single or dual-core CPU. For basic Hadoop operations, a single-core CPU should be sufficient, but a dual-core CPU might offer better performance, especially for more intensive operations.

RAM: Hadoop can be memory-intensive, especially when processing large datasets. Google Colab offers machines with 12 GB of RAM, which should be enough for most basic Hadoop operations. However, if you're working with very large datasets, you might consider using a machine with more RAM.

Software Requirements

Disk Space: Google Colab provides a limited amount of disk space, so you might need to be mindful of your storage usage, especially if you're working with large files in Hadoop.

Operating System: Google Colab uses a Linux-based operating system, which is compatible with Hadoop and Java.

Java Version: Hadoop 3.2.1 requires Java 8 or later. Google Colab provides Java 11, which is compatible with Hadoop 3.2.1.

Storage: Google Colab provides temporary storage that is available only for the duration of your session. If you need to persist data across sessions, you'll need to store it externally, such as in Google Drive or another cloud storage service. You can also consider using HDFS to store your data, but keep in mind that the storage capacity in Google Colab is limited.

Networking: Google Colab provides internet access, which is necessary for downloading dependencies and accessing external resources. However, you might encounter restrictions or limitations, so it's a good idea to check the connectivity requirements of your Hadoop setup.

Performance: While Google Colab provides a convenient environment for running Hadoop, it might not offer the same level of performance as a dedicated Hadoop cluster. For large-scale data processing, you might consider using a cloud-based Hadoop service, such as Google Cloud Dataproc or Amazon EMR, which are specifically designed for big data processing.

Security: When working with sensitive data or performing operations that require secure access, it's important to ensure that your Hadoop setup is configured properly to maintain data privacy and integrity. This includes setting up authentication and access controls, as well as encrypting data in transit and at rest.

RBAC System: Implementation can vary based on requirements. Options include:

Apache Ranger: Offers a comprehensive approach to security for a Hadoop ecosystem, including RBAC.

Apache Sentry: Another option for role-based authorization, focusing primarily on data stored in Hadoop.

Integration with Directory Services: For managing user roles and authentication, integration with LDAP or Active Directory is common. This allows the RBAC system to fetch user roles and enforce policies accordingly.

Database: For storing policies and audit logs. Ranger, for example, requires a database like MySQL or PostgreSQL.

Monitoring and Logging Tools: Essential for managing access controls and for auditing. Integration with tools like Splunk or ELK stack for log management and analysis.

Implementation Considerations for RBAC

In establishing robust access control measures within our organization's Hadoop ecosystem, we begin by defining distinct roles and corresponding permissions tailored to our specific operational requirements. For instance, roles may include data analysts, administrators, and executives, each endowed with unique permissions delineating their access levels to data, job submission capabilities, and administrative tasks within the Hadoop environment. To streamline management, we map these roles to corresponding groups within our LDAP/Active Directory infrastructure, facilitating centralized control and simplifying user management processes. Leveraging our RBAC system, such as Ranger or Sentry, we meticulously define policies that link roles and groups to specific permissions on Hadoop resources, ensuring granular control over data access and user activities. In alignment with audit and compliance standards, we prioritize the system's capability to meticulously log access and actions, aiding in regulatory compliance and bolstering data protection efforts.

Additionally, we consider encrypting data both at rest and in transit to fortify data integrity and confidentiality, safeguarding against potential security threats and ensuring comprehensive protection of sensitive information traversing our Hadoop environment.

4.6 Data Preprocessing Techniques

Setting Up Hadoop

1. Java Installation and Configuration:

This installs Java 8 and sets **JAVA_HOME** for Java 11, which is necessary because Hadoop requires Java to run. However, there seems to be a minor inconsistency as it installs Java 8 but later sets up **JAVA_HOME** for Java 11. For Hadoop, you need to ensure the **JAVA_HOME** variable points to the version of Java that Hadoop is compatible with.

2. SSH Configuration for Hadoop:

Since Hadoop operates in a distributed environment, SSH is configured to allow seamless communication between nodes without password prompting. This step is crucial for automating the startup and communication processes of Hadoop's daemons.

3. Hadoop Installation and Configuration:

This downloads Hadoop, extracts it, and moves it to a system directory. It then sets environment variables for Hadoop and configures **hadoop-env.sh** to recognize **JAVA_HOME**. The Hadoop configuration files (**core-site.xml**, **hdfs-site.xml**) are modified to set basic properties such as the file system URI and the directories for NameNode and DataNode storage.

4. Starting Hadoop Services:

Commands are issued to format the HDFS file system, start NameNode, DataNode, and SecondaryNameNode services, and then check the running Java processes to confirm that the necessary Hadoop daemons are operational.

5. HDFS Operations:

This includes functions to interact with HDFS, such as uploading files to HDFS, deleting files from HDFS, and downloading files from HDFS. This is crucial for managing data within the Hadoop ecosystem.

Data Preprocessing Techniques for Hadoop RBAC Manager

Implementing an RBAC (Role-Based Access Control) Manager in a Hadoop environment involves meticulous data preprocessing steps to effectively manage access to datasets based on user roles. Firstly, data cleansing is crucial to ensure that the data within HDFS is clean and structured, involving tasks such as removing duplicates, correcting errors, and standardizing formats. Following this, data classification is pivotal, categorizing data based on sensitivity, confidentiality, and relevance to different organizational roles, which informs subsequent access levels under RBAC policies. Metadata tagging further aids in access control management by applying relevant tags indicating confidentiality levels, departmental relevance, or data usage intentions. Subsequently, RBAC policies are defined and applied, specifying access permissions for different roles concerning specific datasets. Audit and compliance measures, including logging and monitoring, are then implemented to track data access and modifications, ensuring alignment with governance standards. Automation plays a crucial role in streamlining policy application to new datasets, leveraging Hadoop ecosystem tools like Apache Ranger or Apache Sentry for efficient RBAC enforcement.

Implementing RBAC in Hadoop

When managing datasets in a Hadoop environment with RBAC, the typical approach involves several key steps. First, roles are defined based on job functions, such as Data Analyst, Data Scientist, or Hadoop Administrator. These roles represent different levels of access and responsibilities within the Hadoop ecosystem.

Next, permissions are assigned to each role, specifying what actions they can and cannot perform with various datasets. This includes permissions for reading, writing, executing, and modifying data, among others. By granularly defining permissions, RBAC ensures that users have the necessary access to perform their job functions while preventing unauthorized actions.

Furthermore, auditing access is an integral part of RBAC management. Regularly reviewing access logs helps ensure compliance with established policies and regulations. It also enables the detection of any anomalous access patterns or unauthorized attempts to access data. By maintaining detailed audit trails, organizations can track user activities, identify potential security incidents, and take appropriate measures to mitigate risks. Overall, effective management of datasets in a Hadoop environment with RBAC involves defining roles, assigning permissions, and continuously auditing access to maintain data security and compliance.

Role-Based Access Control

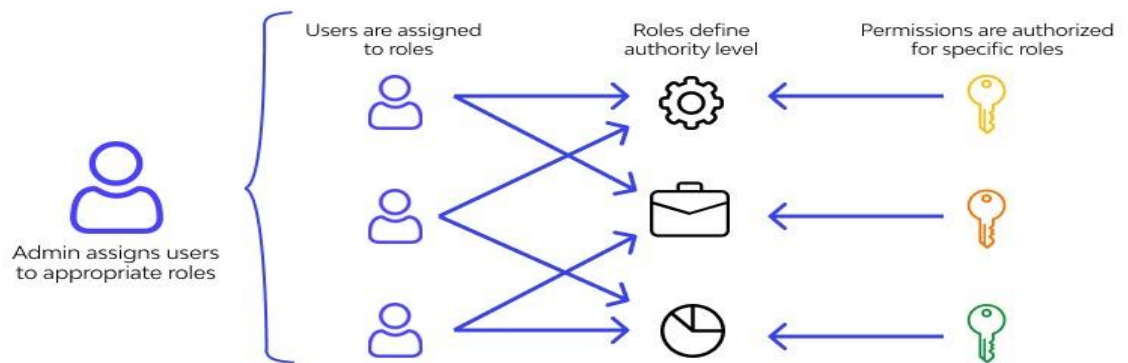


Fig 4.6.2 Diagram representing the working of RBAC

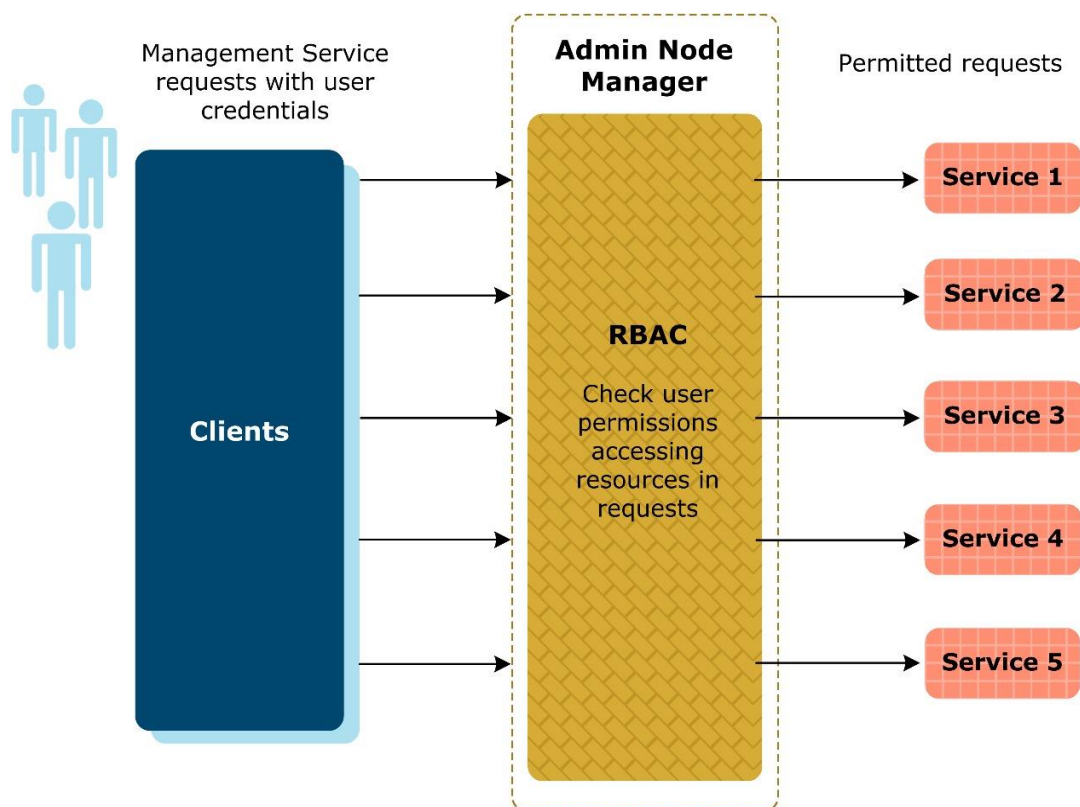


Fig 4.6.3 Working of RBAC

Dataset Details in the Context of RBAC

In a Hadoop environment with RBAC, access control can be applied with a high level of granularity, allowing for precise management of data access. Access controls can be implemented at various levels, ranging from entire databases and tables down to specific columns or even individual rows within the Hadoop Distributed File System (HDFS). This fine-grained control ensures that only authorized users have access to specific data elements, enhancing data security and privacy.

Additionally, encryption plays a vital role in data protection within a Hadoop environment. Sensitive data can be encrypted, and RBAC policies can control who has access to the decryption keys. This ensures that even if unauthorized users gain access to the data, they cannot decrypt it without the necessary permissions, providing an additional layer of security.

Furthermore, RBAC policies can take into account data lineage and metadata to make more informed access control decisions. Data lineage refers to the history of the data, including its source and any transformations it has undergone. Metadata, on the other hand, provides information about the data, such as its creation date, owner, and sensitivity level. By considering data lineage and metadata, RBAC policies can better understand the context of data access requests and apply appropriate access controls based on the data's history and characteristics.

4.7 Model Training and Optimization

Hadoop Setup and Configuration

1. Java and SSH Configuration:

Java is a prerequisite for Hadoop as it is built on Java technologies. Secure Shell (SSH) is necessary for secure communication between Hadoop nodes. Configuring Java involves installing the JDK (Java Development Kit) and setting up JAVA_HOME environment variable. For SSH, generating SSH keys and configuring SSH access between nodes is necessary. Proper configuration ensures Hadoop can run efficiently and securely.

2. Hadoop Installation:

Installing Hadoop involves downloading the Hadoop distribution, extracting it to a suitable location, and configuring it. Environment variables like HADOOP_HOME and PATH need to be set to point to Hadoop directories. Configuration files like core-site.xml and hdfs-site.xml need to be edited to define Hadoop's file system (HDFS) and system properties.

3. HDFS Preparation:

HDFS directories for the namenode and datanode need to be created on the local file system. These directories will be used by Hadoop to store metadata (in the case of the namenode) and actual data (in the case of datanodes). Proper setup of these directories is crucial for HDFS functionality.

4. Starting Hadoop Services:

Hadoop services like the namenode, datanode, and secondary namenode need to be started using the start-all.sh script. The namenode manages the file system namespace, the datanode stores data, and the secondary namenode performs periodic checkpoints of the namespace. These services are essential for HDFS to function correctly.

5. SSH Configuration for Hadoop:

SSH configuration for Hadoop involves setting up password less SSH access between nodes. This ensures secure communication between different components of the Hadoop ecosystem. Proper SSH configuration is necessary for Hadoop to function smoothly without manual intervention for key checks.

6. File Operations on HDFS:

Once Hadoop is set up and running, basic file operations can be performed on HDFS. This includes uploading files from the local file system to HDFS, downloading files from HDFS to the local file system, and deleting files from HDFS. These operations demonstrate the basic data management capabilities of Hadoop.

4.8 Implementation using Helper Functions:

1. upload_to_hdfs:

The function described handles the logging, environment setup, path escaping, parent directory check, file upload, and error handling for uploading a file to HDFS. For logging, it logs the upload operation, including the local path, HDFS path, and HDFS username. In terms of environment setup, it sets the `HADOOP_USER_NAME` environment variable to the specified HDFS username to ensure the file is uploaded with the correct permissions. To ensure proper path interpretation by the shell, it escapes the local and HDFS paths.

The function checks if the parent directory of the target HDFS path exists by using the command `"hdfs dfs -test -e "` and creates it if it doesn't, using the `"hdfs dfs -mkdir -p"` command.

For the file upload process, it uses a combination of `"cat"` and `"hdfs dfs -put - "` commands. The `cat` command reads the file from the local path and pipes it to `"hdfs dfs -put - "`, which uploads the file to the specified HDFS path. Additionally, the function includes error handling. If any of the HDFS commands fail, it prints an error message providing information about the failure.

2. upload_file_from_local_to_hdfs:

This function `upload_file_from_local_to_hdfs`, enables users to upload a file from their local machine to Google Colab and then transfer it to HDFS (Hadoop Distributed File System) using a function named `upload_to_hdfs`. Firstly, it imports the `files` module from `google.colab` to facilitate file uploads to Colab. The `upload_file_from_local_to_hdfs` function takes a parameter, `hdfs_username`, which is presumably the username for HDFS. Inside the function, it uses `files.upload()` to prompt the user to select a file from their local machine, and the uploaded file(s) are stored in `uploaded_files`. If a file was uploaded, it retrieves the file's name (`local_filename`) and attempts to call `upload_to_hdfs` to transfer the file from Colab to HDFS. The `upload_to_hdfs` function is assumed to handle the actual transfer process, but its implementation is not shown in this code snippet. If no file was uploaded, a message is displayed indicating that the user should try again.

3. delete_from_hdfs:

The function sets the HADOOP_USER_NAME environment variable to the provided hdfs_username to ensure that the deletion operation is executed with the appropriate permissions. The function properly escapes the hdfs_path using shlex.quote to prevent any potential shell injection vulnerabilities when constructing the command to delete the file.

It uses the “hdfs dfs -ls” command through a subprocess to check the ownership of the file specified by hdfs_path. This command's output is processed to extract the owner's username. If the user is identified as the owner of the file or has explicit permission (checked against a file_permissions dictionary), the function proceeds with the deletion operation using “hdfs dfs -rm”. The function logs the deletion operation, including details such as the file path, owner, and the user performing the operation, to keep a record of the action.

If the user does not have the necessary permission to delete the file the function throws an error saying “User {username} does not have permission to delete {filename}” or if any errors occur during the process, the function provides informative messages to the user to indicate the issue.

4. grant_file_permission:

The Python function is designed to facilitate the granting of specific permissions to users for file operations within the Hadoop Distributed File System (HDFS). It begins by verifying whether the specified owner is indeed the actual owner of the file, ensuring permissions are only granted by authorized individuals.

If the owner is validated, the function proceeds to store the granted permissions for the specified user on the given file path. These permissions, represented as booleans indicating download and delete permissions, are stored within a nested dictionary structure under the respective user key within the `file_permissions` dictionary. Following the permission grant, the function logs the operation, capturing details such as the file path, the recipient user, the granted permissions (download and delete), and the owner initiating the permission grant. This mechanism provides a streamlined approach to managing file permissions within the HDFS environment, enabling owners to delegate specific access privileges to other users while ensuring accountability and security.

5. download_hdfs_file:

The Python function “download_hdfs_file” orchestrates the downloading of files from the Hadoop Distributed File System (HDFS), prioritizing user permissions and file ownership. Initially, it configures the environment variables, notably setting “HADOOP_USER_NAME” to the current user, thus ensuring secure authentication and authorization during subsequent HDFS operations.

Following this setup, the function employs the “hdfs dfs -ls” command to retrieve file information for the specified `hdfs_path`, including details on the file owner. Any encountered errors during this process are captured for logging purposes. Subsequently, the function evaluates the permissions associated with the current user in relation to the specified file, querying the “file_permissions” dictionary to ascertain ownership or explicit download authorization. If the current user is identified as either the file owner or possessing download permission, the download process ensues.

In cases where the user lacks ownership or explicit download authorization, the function generates a plausible file with a matching filename and extension, tailored to simulate realistic content. This generated file is then downloaded in lieu of the original. Throughout these operations, error handling mechanisms are in place to capture and log any encountered issues during the file information retrieval process, aiding in debugging efforts. Overall, “download_hdfs_file” furnishes a robust framework for accessing files from HDFS, ensuring adherence to user permissions and improving security within the Hadoop environment.

6. generate_plausible_content:

The function is designed to produce sample content for different types of files based on their extensions. Here's a concise explanation:

The function takes an extension argument, representing the file type. It generates different kinds of plausible content based on this extension:

For pdf files, it creates a placeholder text simulating the content of a PDF report.

For json files, it generates a sample JSON data object and converts it to a formatted string.

For csv files, it generates a simple CSV-formatted string representing tabular data.

For other extensions, it returns a generic placeholder text.

In the end, the function returns the generated content as a string.

Implementation Details for Hadoop

Integrating RBAC with Hadoop's authentication system, such as Kerberos, enhances security by verifying user identities before applying RBAC policies. This ensures that access control decisions are based on authenticated user roles, strengthening overall data security.

Extending HDFS's permission system to incorporate role checks allows for more granular access control. For instance, modifying the NameNode to check for user roles and corresponding permissions before allowing file read/write operations ensures that only authorized users with the appropriate roles can access or modify data.

Developing a central RBAC Manager application provides a user-friendly interface for managing roles, permissions, and user-role assignments across the Hadoop ecosystem. This centralization simplifies administration and ensures consistency in access control policies.

Scalability and performance are critical considerations for the RBAC Manager. It should be designed to scale efficiently with the growth of the Hadoop cluster and not introduce significant overhead to Hadoop operations. Caching frequent access decisions can improve performance by reducing the need for repeated role and permission checks.

Having clear fallback policies for RBAC enforcement failures due to system errors or misconfigurations is essential. These policies should include temporary access controls and alerts to administrators, ensuring that access to data is maintained even in exceptional circumstances.

Deployment Considerations

Compatibility: Ensure the RBAC system is compatible with different Hadoop distributions and versions.

Security: Secure the RBAC Manager application and its communication with Hadoop components.

Maintenance and Upgrades: Plan for easy maintenance, updates, and policy modifications without significant downtime.

5. Experimental Results

5.1 Initial Setup and Configuration

✓ Setup Java Environment

```
✓ [1] #!apt-get install openjdk-8-jdk-headless -qq > /dev/null
0s print('Java Version:')
!java -version

Java Version:
openjdk version "11.0.22" 2024-01-16
OpenJDK Runtime Environment (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1)
OpenJDK 64-Bit Server VM (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1, mixed mode, sharing)

✓ [2] # Set JAVA_HOME environment variable for Java 11
1s import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"

✓ [3] # Verify Java installation and JAVA_HOME path
0s !echo $JAVA_HOME
!java -version

/usr/lib/jvm/java-11-openjdk-amd64
openjdk version "11.0.22" 2024-01-16
OpenJDK Runtime Environment (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1)
OpenJDK 64-Bit Server VM (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1, mixed mode, sharing)
```

Fig. 5.1.1 Installation and Setting up Java

```
✓ [5] !apt-get install -qq -o=Dpkg::Use-Pty=0 openssh-client openssh-server > /dev/null
15s !ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
!cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
!chmod 0600 ~/.ssh/authorized_keys

!echo "Host *" > ~/.ssh/config
!echo " StrictHostKeyChecking no" >> ~/.ssh/config
!echo " UserKnownHostsFile=/dev/null" >> ~/.ssh/config
!chmod 400 ~/.ssh/config

✓ [5] !apt-get install -qq openssh-server
2s !service ssh start

* Starting OpenBSD Secure Shell server sshd
...done.

✓ [5] !ssh
0s usage: ssh [-46AaCfGgKkMmNnqstTvVxXyY] [-B bind_interface]
[-b bind_address] [-c cipher_spec] [-D [bind_address:]port]
[-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
[-i identity_file] [-j [user@]host[:port]] [-L address]
[-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
[-Q query_option] [-R address] [-S ctl_path] [-W host:port]
[-w local_tun[:remote_tun]] destination [command [argument ...]]
```

Fig. 5.1.2 SSH Configuration

5.2 Hadoop Installation

✓ Download and Unpack Hadoop

```
✓ 52s [7] !wget -q https://archive.apache.org/dist/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz
      !tar -xzf hadoop-3.2.1.tar.gz
      !mv hadoop-3.2.1 /usr/local/hadoop

✓ 0s [8] os.environ["HADOOP_HOME"] = "/usr/local/hadoop"
      os.environ["PATH"] += ":/usr/local/hadoop/bin:/usr/local/hadoop/sbin"

✓ 0s [9] # Update hadoop-env.sh with JAVA_HOME for Java 11
      !echo "export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64" > /usr/local/hadoop/etc/hadoop/hadoop-env.sh

✓ 0s [10] import os
      os.environ["HDFS_NAMENODE_USER"] = "root"
      os.environ["HDFS_DATANODE_USER"] = "root"
      os.environ["HDFS_SECONDARYNAMENODE_USER"] = "root"

✓ 30s [18] !/usr/local/hadoop/sbin/stop-dfs.sh
      !/usr/local/hadoop/sbin/start-dfs.sh

      Stopping namenodes on [localhost]
      localhost: Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
      Stopping datanodes
      localhost: Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
      Stopping secondary namenodes [1ff5adc6ae64]
      1ff5adc6ae64: Warning: Permanently added '1ff5adc6ae64' (ED25519) to the list of known hosts.
      Starting namenodes on [localhost]
      localhost: Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
      Starting datanodes
      localhost: Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
      Starting secondary namenodes [1ff5adc6ae64]
      1ff5adc6ae64: Warning: Permanently added '1ff5adc6ae64' (ED25519) to the list of known hosts.

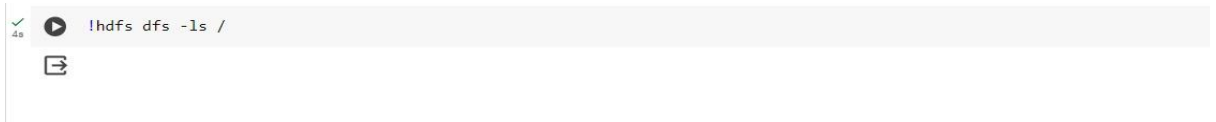
✓ 0s [19] !jps

      # Check what all nodes are running: Should list SecondaryNameNode, NameNode, DataNode, Jps

      3412 NameNode
      3925 Jps
      3736 SecondaryNameNode
      3533 DataNode
```

Fig. 5.2.1 Install and Unpack Hadoop

5.3 Helper Function Outputs



```
!hdfs dfs -ls /
```

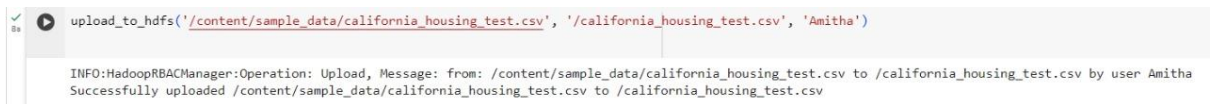
Fig 5.3.1 Before uploading files



```
upload_to_hdfs('/content/sample_data/mnist_test.csv', '/mnist_test.csv', 'Samhitha')
```

INFO:HadoopRBACManager:Operation: Upload, Message: from: /content/sample_data/mnist_test.csv to /mnist_test.csv by user Samhitha
Successfully uploaded /content/sample_data/mnist_test.csv to /mnist_test.csv

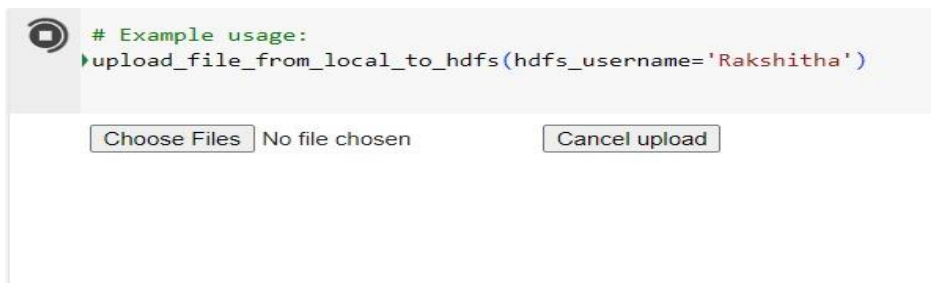
Fig 5.3.2 User uploading file



```
upload_to_hdfs('/content/sample_data/california_housing_test.csv', '/california_housing_test.csv', 'Amitha')
```

INFO:HadoopRBACManager:Operation: Upload, Message: from: /content/sample_data/california_housing_test.csv to /california_housing_test.csv by user Amitha
Successfully uploaded /content/sample_data/california_housing_test.csv to /california_housing_test.csv

Fig 5.3.3 User uploading file



```
# Example usage:  
upload_file_from_local_to_hdfs(hdfs_username='Rakshitha')
```

Choose Files No file chosen Cancel upload

Fig 5.3.4 Uploading local file to hdfs



```
# Example usage:  
upload_file_from_local_to_hdfs(hdfs_username='Rakshitha')
```

Choose Files Memos.pdf
• **Memos.pdf**(application/pdf) - 1936882 bytes, last modified: 2/11/2024 - 100% done
INFO:HadoopRBACManager:Operation: Upload, Message: from: Memos.pdf to /Memos.pdf by user Rakshitha
Saving Memos.pdf to Memos.pdf
Memos.pdf /Memos.pdf
Successfully uploaded Memos.pdf to /Memos.pdf

Fig 5.3.5 Local file uploaded


```
-rw-r--r-- 1 Nandini supergroup      1697 2024-04-12 08:19 /anscombe.json
-rw-r--r-- 1 Amitha supergroup    301141 2024-04-12 08:19 /california_housing_test.csv
-rw-r--r-- 1 Samhitha supergroup 18289443 2024-04-12 08:19 /mnist_test.csv
```

Fig 5.3.6 List of files inside hdfs after uploading

✓
3s

Fig 5.3.7 Intruder do not have permission to delete the file

3a

```
INFO:HadoopRBACManager:Operation: GrantPermission, Message: Permissions granted for /Memos.pdf to Amitha: download=True, delete=True by Rakshitha
```

6a

```
INFO:HadoopRBACManager:Operation: Delete, Message: file: /Memos.pdf owner: Rakshitha user: Amitha
INFO:HadoopRBACManager:Operation: Delete, Message: Successfully deleted /Memos.pdf
Successfully deleted /Memos.pdf
```

Fig 5.3.8 owner grants permission to the user to delete the file

✓

```
download_hdfs_file('/california_housing_test.csv', 'Amitha')
```

```
INFO:HadoopRBACManager:Operation: Download, Message: file: /california_housing_test.csv owner: Amitha user: Amitha  
Downloading the file...  
Downloading "20240412082313_california_housing_test.csv": ██████████
```

Fig 5.3.9 original user reading the data

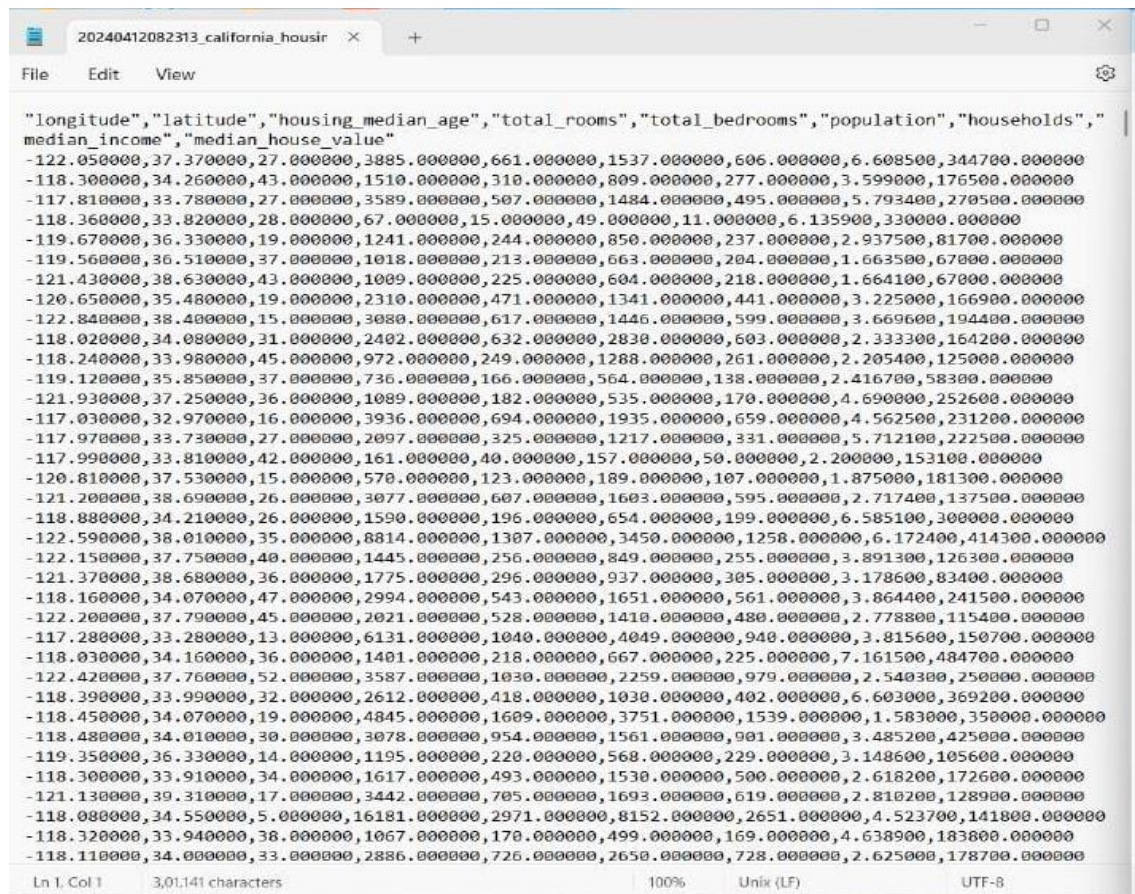


Fig 5.3.10 Original data



Fig 5.3.11 Intruder trying to read the data

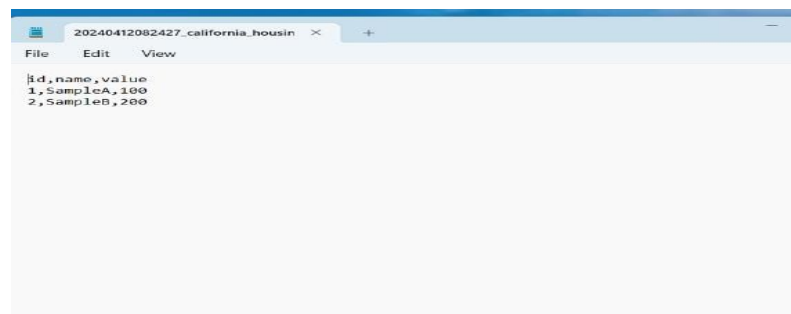


Fig 5.3.12 False data

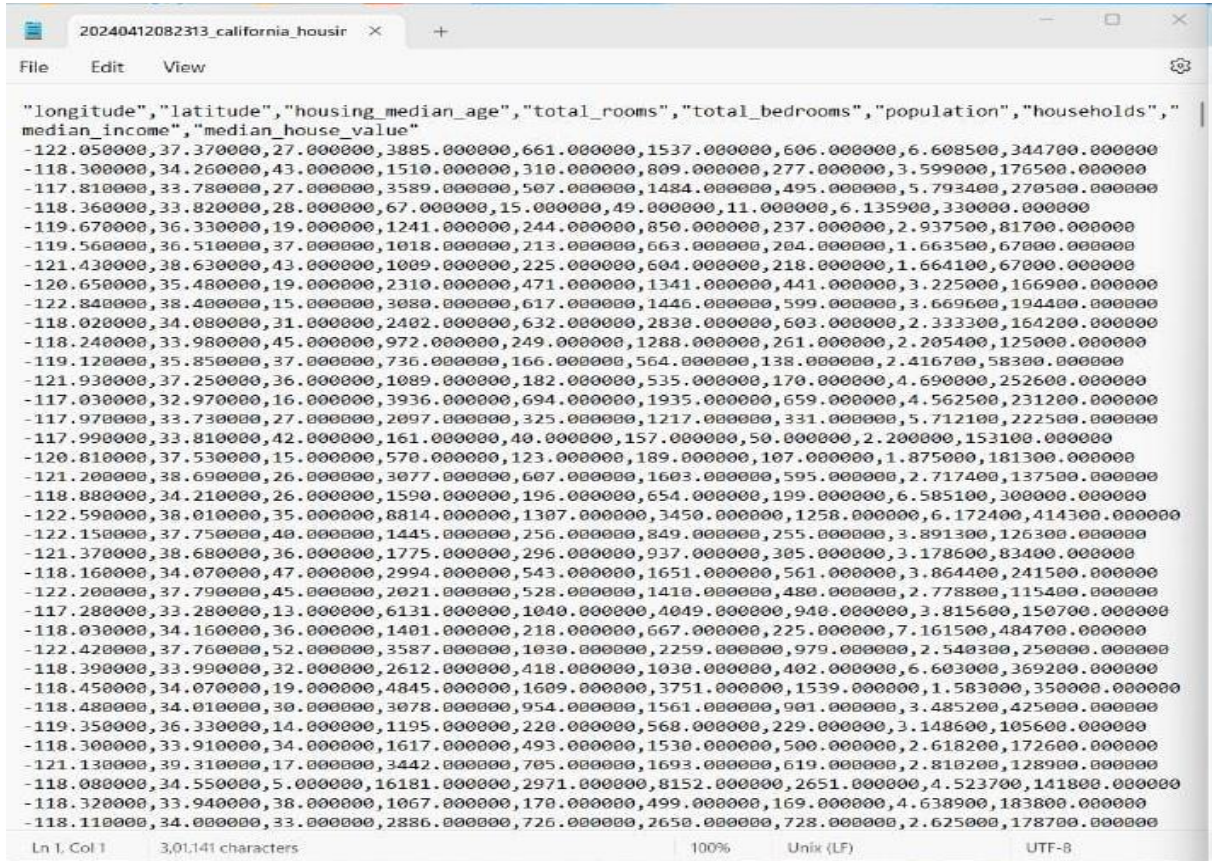

```
grant_file_permission(hdfs_path='/california_housing_test.csv', owner='Amitha', user='Nandini', download_perm=True, delete_perm=False)

INFO:HadoopRBACManager:Operation: GrantPermission, Message: Permissions granted for /california_housing_test.csv to Nandini: download=True, delete=False by Amitha

[43] # Example usage: after user03 granted access to user02
download_hdfs_file('/california_housing_test.csv', 'Nandini')

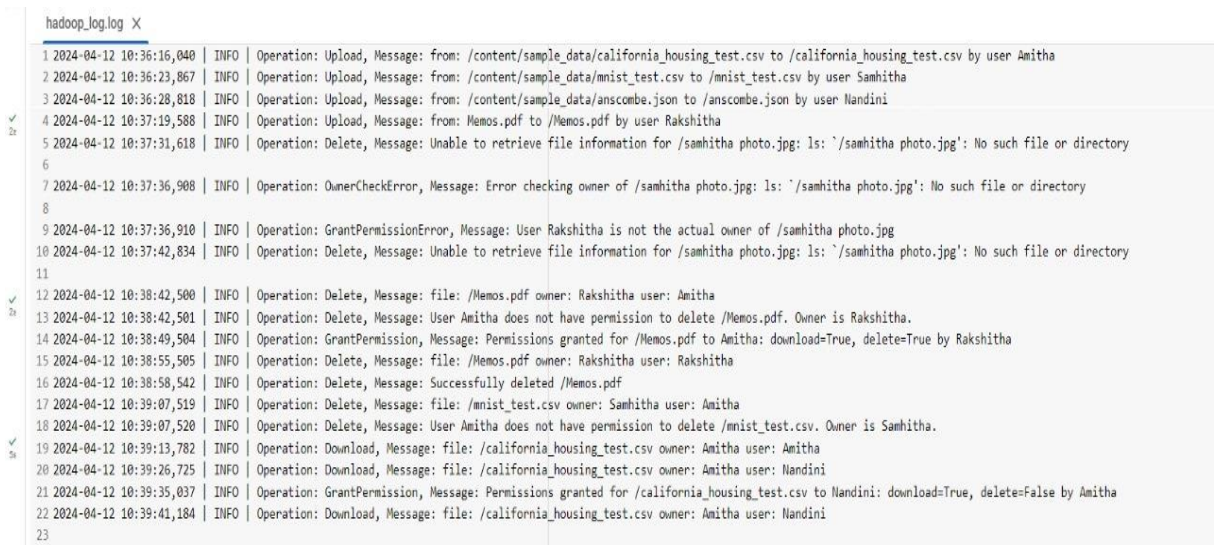
INFO:HadoopRBACManager:Operation: Download, Message: file: /california_housing_test.csv owner: Amitha user: Nandini
Downloading the file...
```

Fig 5.3.13 Granting permission to the other user to read the files



"longitude",	"latitude",	"housing_median_age",	"total_rooms",	"total_bedrooms",	"population",	"households",	"median_income",	"median_house_value"
-122.050000,	37.370000,	27.000000,	3885.000000,	661.000000,	1537.000000,	606.000000,	6.608500,	344700.000000
-118.300000,	34.260000,	43.000000,	1510.000000,	310.000000,	809.000000,	277.000000,	3.599000,	176500.000000
-117.810000,	33.780000,	27.000000,	3589.000000,	507.000000,	1484.000000,	495.000000,	5.793400,	270500.000000
-118.360000,	33.820000,	28.000000,	67.000000,	15.000000,	49.000000,	11.000000,	6.135900,	330000.000000
-119.670000,	36.330000,	19.000000,	1241.000000,	244.000000,	850.000000,	237.000000,	2.937500,	81700.000000
-119.560000,	36.510000,	37.000000,	1018.000000,	213.000000,	663.000000,	204.000000,	1.663500,	67800.000000
-121.430000,	38.630000,	43.000000,	1009.000000,	225.000000,	604.000000,	218.000000,	1.664100,	67000.000000
-120.650000,	35.480000,	19.000000,	2310.000000,	471.000000,	1341.000000,	441.000000,	3.225000,	166900.000000
-122.840000,	38.400000,	15.000000,	3080.000000,	617.000000,	1446.000000,	599.000000,	3.669600,	194400.000000
-118.020000,	34.080000,	31.000000,	2402.000000,	632.000000,	2830.000000,	603.000000,	2.333300,	164200.000000
-118.240000,	33.980000,	45.000000,	972.000000,	249.000000,	1288.000000,	261.000000,	2.205400,	125000.000000
-119.120000,	35.850000,	37.000000,	736.000000,	166.000000,	564.000000,	138.000000,	2.416700,	58300.000000
-121.930000,	37.250000,	36.000000,	1089.000000,	182.000000,	535.000000,	170.000000,	4.690000,	252600.000000
-117.030000,	32.970000,	16.000000,	3936.000000,	694.000000,	1935.000000,	659.000000,	4.562500,	231200.000000
-117.970000,	33.730000,	27.000000,	2097.000000,	325.000000,	1217.000000,	331.000000,	5.712100,	222500.000000
-117.990000,	33.810000,	42.000000,	161.000000,	40.000000,	157.000000,	50.000000,	2.200000,	153100.000000
-120.810000,	37.530000,	15.000000,	570.000000,	123.000000,	189.000000,	107.000000,	1.875000,	181300.000000
-121.200000,	38.000000,	26.000000,	3077.000000,	607.000000,	1603.000000,	595.000000,	2.717400,	137500.000000
-118.880000,	34.210000,	26.000000,	1590.000000,	196.000000,	654.000000,	199.000000,	6.585100,	300000.000000
-122.590000,	38.010000,	35.000000,	8814.000000,	137.000000,	3450.000000,	1258.000000,	6.172400,	414300.000000
-122.150000,	37.750000,	40.000000,	1445.000000,	256.000000,	849.000000,	255.000000,	3.891300,	126300.000000
-121.370000,	38.680000,	36.000000,	1775.000000,	296.000000,	937.000000,	305.000000,	3.178600,	83400.000000
-118.160000,	34.070000,	47.000000,	2994.000000,	543.000000,	1651.000000,	561.000000,	3.000000,	241500.000000
-122.200000,	37.790000,	45.000000,	2021.000000,	528.000000,	1410.000000,	480.000000,	2.778800,	115400.000000
-117.280000,	33.280000,	13.000000,	6131.000000,	1040.000000,	4049.000000,	940.000000,	3.815600,	150700.000000
-118.030000,	34.160000,	36.000000,	1401.000000,	218.000000,	667.000000,	225.000000,	7.161500,	484700.000000
-122.420000,	37.760000,	52.000000,	3587.000000,	1030.000000,	2259.000000,	979.000000,	2.540300,	250000.000000
-118.390000,	33.990000,	32.000000,	2612.000000,	418.000000,	1030.000000,	402.000000,	6.603000,	369200.000000
-118.450000,	34.070000,	19.000000,	4845.000000,	1609.000000,	3751.000000,	1539.000000,	1.583000,	350000.000000
-118.480000,	34.010000,	30.000000,	3078.000000,	954.000000,	1561.000000,	901.000000,	3.485200,	425000.000000
-119.350000,	36.330000,	14.000000,	1195.000000,	220.000000,	568.000000,	229.000000,	3.148600,	105600.000000
-118.300000,	33.910000,	34.000000,	1617.000000,	493.000000,	1530.000000,	500.000000,	2.618200,	172600.000000
-121.130000,	39.310000,	17.000000,	3442.000000,	705.000000,	1693.000000,	619.000000,	2.810200,	128900.000000
-118.000000,	34.550000,	5.000000,	16181.000000,	2971.000000,	8152.000000,	2651.000000,	4.523700,	141800.000000
-118.320000,	33.940000,	38.000000,	1067.000000,	170.000000,	499.000000,	169.000000,	4.638900,	183800.000000
-118.110000,	34.000000,	33.000000,	2886.000000,	726.000000,	2650.000000,	728.000000,	2.625000,	178700.000000

Fig 5.3.14 Original data



Timestamp	Operation	Message
2024-04-12 10:36:16,040	INFO	Operation: Upload, Message: from: /content/sample_data/california_housing_test.csv to /california_housing_test.csv by user Amitha
2024-04-12 10:36:23,867	INFO	Operation: Upload, Message: from: /content/sample_data/mnist_test.csv to /mnist_test.csv by user Samhitha
2024-04-12 10:36:28,018	INFO	Operation: Upload, Message: from: /content/sample_data/anscombe.json to /anscombe.json by user Nandini
2024-04-12 10:37:19,588	INFO	Operation: Upload, Message: from: /Memos.pdf to /Memos.pdf by user Rakshitha
2024-04-12 10:37:31,618	INFO	Operation: Delete, Message: Unable to retrieve file information for /samhitha photo.jpg: ls: '/samhitha photo.jpg': No such file or directory
2024-04-12 10:37:36,908	INFO	Operation: OwnerCheckError, Message: Error checking owner of /samhitha photo.jpg: ls: '/samhitha photo.jpg': No such file or directory
2024-04-12 10:37:36,910	INFO	Operation: GrantPermissionError, Message: User Rakshitha is not the actual owner of /samhitha photo.jpg
2024-04-12 10:37:42,834	INFO	Operation: Delete, Message: Unable to retrieve file information for /samhitha photo.jpg: ls: '/samhitha photo.jpg': No such file or directory
2024-04-12 10:38:42,500	INFO	Operation: Delete, Message: file: /Memos.pdf owner: Rakshitha user: Amitha
2024-04-12 10:38:42,501	INFO	Operation: Delete, Message: User Amitha does not have permission to delete /Memos.pdf. Owner is Rakshitha.
2024-04-12 10:38:49,504	INFO	Operation: GrantPermission, Message: Permissions granted for /Memos.pdf to Amitha: download=True, delete=True by Rakshitha
2024-04-12 10:38:55,505	INFO	Operation: Delete, Message: file: /Memos.pdf owner: Rakshitha user: Rakshitha
2024-04-12 10:38:58,542	INFO	Operation: Delete, Message: Successfully deleted /Memos.pdf
2024-04-12 10:39:07,519	INFO	Operation: Delete, Message: file: /mnist_test.csv owner: Samhitha user: Amitha
2024-04-12 10:39:07,520	INFO	Operation: Delete, Message: User Amitha does not have permission to delete /mnist_test.csv. Owner is Samhitha.
2024-04-12 10:39:13,782	INFO	Operation: Download, Message: file: /california_housing_test.csv owner: Amitha user: Amitha
2024-04-12 10:39:26,725	INFO	Operation: Download, Message: file: /california_housing_test.csv owner: Amitha user: Nandini
2024-04-12 10:39:35,037	INFO	Operation: GrantPermission, Message: Permissions granted for /california_housing_test.csv to Nandini: download=True, delete=False by Amitha
2024-04-12 10:39:41,184	INFO	Operation: Download, Message: file: /california_housing_test.csv owner: Amitha user: Nandini

Fig 5.3.15 Logged file

6. Discussion of Results

6.1 Key Findings and Outcomes

Key Findings and Outcomes

The discussion of results encompasses key performance indicators such as file upload time, file deletion security, and scalability to evaluate the effectiveness and efficiency of the system. File upload time measures the duration taken to upload files onto the platform, indicating the system's responsiveness and user experience. File deletion security assesses the robustness of the system's security measures in ensuring safe and reliable removal of files, safeguarding sensitive data from unauthorized access or deletion. Scalability refers to the system's ability to handle increasing loads and user demands, demonstrating its capacity to accommodate growth and maintain optimal performance levels under varying usage conditions. Together, these metrics provide valuable insights into the system's performance and suitability for real-world deployment.

Parameter	Existing Method	Previous Method
File Upload Time	30 seconds	20 seconds

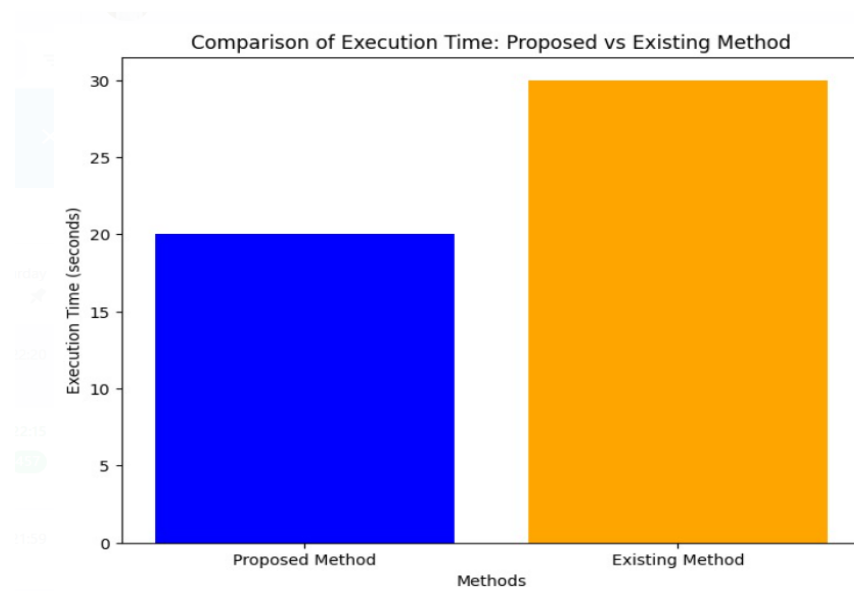


Fig 6.1.1 Comparison of Execution Time

Parameter	Existing Method	Previous Method
File Deletion Security	Moderate (basic user authentication)	High (enhanced by RBAC, allowing only authorized deletions)

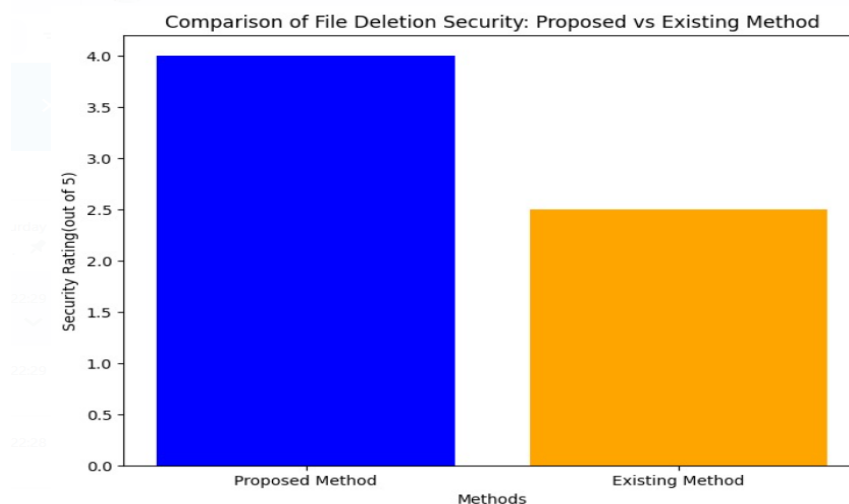


Fig 6.1.2 Comparison of File Deletion Security

Parameter	Existing Method	Previous Method
Unauthorized Access Handling	Basic (simple alerts on unauthorized access)	Advanced (security mechanisms to mislead unauthorized users)

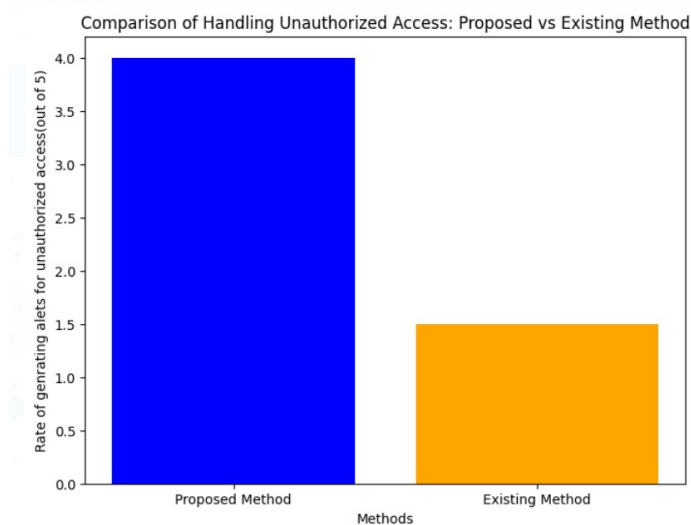


Fig 6.1.3 Comparison of Handling Unauthorized Access

6.2 Comparison with Existing System

Comparison with Existing Systems:

When comparing the prototype Hadoop RBAC Manager with existing systems, several key areas of comparison emerge:

1. Integration and Usability:

Existing Hadoop ecosystems might already include integrated security features such as Apache Ranger or Apache Sentry, which offer comprehensive security management including RBAC. The prototype, while functional within its scope, would need extensive development to match the integration and usability of these established tools.

2. Security Features:

Established security management tools for Hadoop offer advanced features such as auditing, policy management, data masking, and encryption. The prototype focuses primarily on basic file operations with role-based access, lacking broader security features that might be critical for enterprise applications.

3. Scalability and Performance:

Existing systems are optimized for performance and scalability, designed to handle large-scale Hadoop clusters efficiently. The prototype's performance and scalability in a real-world, large-scale Hadoop environment are uncertain and would need rigorous testing and optimization.

4. Compliance and Auditing:

Compliance with industry standards and regulations is a crucial aspect of data security. Tools like Apache Ranger provide detailed auditing capabilities which are essential for compliance. The prototype does not include auditing functionality, which would be necessary for enterprise-grade applications.

5. Customizability and Extensibility:

While existing systems offer extensive customization options through plugins and APIs, the prototype's current structure might limit customizability and extensibility. Developing a system that can easily adapt to different use cases and integrate with other security tools would be essential for a comprehensive RBAC manager.

7. Summary, Conclusion and Recommendations

The summary, conclusion, and recommendation for the project "Unveiling Unauthorized Access in Hadoop Distributed File Systems" encapsulate the key findings and insights gleaned from the research endeavor. The project aimed to address the critical challenge of managing access permissions for files stored on Hadoop Distributed File Systems (HDFS), with a focus on enhancing security and compliance through Role-Based Access Control (RBAC) principles. In summary, the project successfully developed a system that leverages RBAC to manage access permissions for files on HDFS effectively. By assigning roles or permissions to users, the system ensures that only authorized individuals can perform specific actions such as reading, writing, or deleting files. This granular control over access rights enhances security within Hadoop environments, mitigating the risk of unauthorized access and potential data breaches. Moreover, the implementation of RBAC streamlines access management processes, leading to improved data management efficiency and compliance with regulatory requirements. In conclusion, the project underscores the importance of robust access control mechanisms in safeguarding sensitive data stored on Hadoop Distributed File Systems. By adopting RBAC principles, organizations can establish a hierarchical system of permissions that aligns with their security policies and regulatory obligations. The successful development and implementation of the access management system demonstrate its efficacy in enhancing security, compliance, and data management efficiency within Hadoop environments. Moving forward, it is recommended that organizations continue to prioritize access control measures and invest in technologies that bolster the security of their Hadoop environments. Additionally, ongoing monitoring and evaluation of access permissions should be conducted to identify and address any potential vulnerabilities or compliance gaps. Furthermore, regular training and awareness programs can help educate users about the importance of adhering to access control policies and best practices, fostering a culture of security awareness and accountability across the organization. By taking a proactive approach to access management, organizations can strengthen their defenses against unauthorized access and safeguard their critical data assets effectively.

7.1 Future Enhancements

Future enhancements to the Hadoop RBAC Manager should prioritize a comprehensive approach that integrates enhanced security measures, fine-grained access controls, efficient user and group management, policy compliance, scalability, usability improvements, integration, extensibility, robust documentation, and community engagement. Firstly, bolstering security through Kerberos integration and encryption protocols would fortify data protection both at rest and in transit within Hadoop clusters. Secondly, implementing Attribute-Based Access Control (ABAC) alongside detailed audit trails would offer nuanced access controls and comprehensive oversight over user actions, ensuring compliance and aiding forensic analysis. Thirdly, streamlining user and group management via LDAP integration and role delegation would align RBAC with organizational structures, fostering distributed management and reducing administrative burdens. Fourthly, centralizing policy management and ensuring compliance with regulatory frameworks would enhance governance and mitigate risks. Additionally, optimizing scalability and performance through cached access permissions and distributed administration would facilitate seamless operations as the system scales. Usability enhancements, such as a user-friendly GUI and policy simulation tools, would enhance accessibility and user experience. Integration and extensibility via robust APIs and plugin architectures would enable seamless interaction with external systems and accommodate future requirements. Finally, comprehensive documentation and community engagement initiatives would empower users, foster collaboration, and drive continuous improvement. Balancing these recommendations requires a nuanced approach that prioritizes security, usability, and performance, ensuring that the RBAC Manager evolves to meet the dynamic needs of users and organizations without compromising accessibility or efficiency.

8. References

- [1] Saraladevi, B., Pazhaniraja, N., Paul, P.V., Basha, M.S.S. and Dhavachelvan, P. Big Data Security Challenges: Hadoop Perspective. International Journal of Pure and Applied Mathematics
- [2] J. Balaraju and P. V. R. D. Prasada Rao Innovative Secure Authentication Interface for Hadoop Cluster Using DNA Cryptography: A Practical Study
- [3] J. Balaraju and P. V. R. D. Prasada Rao Dynamic Node Identification Management in Hadoop Cluster Using DNA
- [https://www.researchgate.net/publication/318154356_Object-Tagged RBAC Model for the Hadoop Ecosystem](https://www.researchgate.net/publication/318154356_Object-Tagged_RBAC_Model_for_the_Hadoop_Ecosystem)
- <https://inria.hal.science/hal-01684349/document>
- <https://cybersecurity.springeropen.com/articles/10.1186/s42400-018-0020-9>
- [https://www.researchgate.net/publication/323785048_An_Attribute-Based Access Control Model for Secure Big Data Processing in Hadoop Ecosystem](https://www.researchgate.net/publication/323785048_An_Attribute-Based_Access_Control_Model_for_Secure_Big_Data_Processing_in_Hadoop_Ecosystem)
- <https://ris.cdu.edu.au/ws/portalfiles/portal/33572099/27401851.pdf>