IDify - Distributed Database System for Digital Identification

Tishan Wickramasinghe, Viraj Pathirage, Chathraka Ranaweera, Rangana Chandrasekara
Dr. Bhagya Nathali Silva, Ms. Dilani Ranaweera
Department of Computer Engineering
Faculty of Engineering
University of Sri Jayewardenepura

***Abstract*—"IDify" Distributed database system for digital identification is a big data handling system to centralize the important services provided by the government to make the facilities efficient and reliable. It has integrated services such as digital authentication, E-Banking, Personal Information Management, and Government Surveys to provide users with a centralized and digitized platform to make their lives easier. In this system there is a distributed database platform to carry out all the data storing and data processing tasks to facilitate a large user count and to avoid the drawbacks of monolithic systems. The database system has been designed by integrating several modern database technologies to make the system efficient and coherent as the facilities provided by the system are extremely mission-critical. To make the system low latent on data ingestion and data reading tasks a special library is also implemented in collaboration with modern query processing engines. It is proposed to utilize the thread system of a processor to implement parallelism to enhance the performance of data read/write tasks. Apart from that a fault-tolerant framework has been implemented to free the system from any erroneous scenario taking place.**

***Index Terms*—Distributed Database System, KUDU, HDFS, HBase, Impala**

I. Introduction

A distributed system is a computing environment in which various components are spread across multiple computing devices on a network. These devices split up the work, coordinating their efforts to complete the job more efficiently than if a single device had been responsible for the task.

Distributed systems are an important development for IT and computer science as an increasing number of related jobs are so massive and complex that it would be impossible for a single computer to handle them alone. Distributed systems reduce the risks involved with having a single point of failure, bolstering reliability and fault tolerance. Modern distributed systems are generally designed to be scalable in near real-time; also, you can spin up additional computing resources on the fly, increasing performance and further reducing time to completion. Distributed systems offer many advantages over monolithic, or single, systems, including greater flexibility, reliability, enhanced speed, security, and many more. Under reliability, it means the ability to withstand failures in one or more of its nodes without severely impacting performance. In a monolithic system, the entire application goes down if the server goes down. Apart from that heavy traffic can bog down single servers when traffic gets heavy, impacting performance for everyone. The scalability of distributed databases and other distributed systems makes them easier to maintain and also sustain high-performance levels.

Distributed systems are considerably more complex than monolithic computing environments, and raise many challenges around design, operations, and maintenance. These include Increased opportunities for failure, synchronization process challenges, imperfect scalability, and complexity. The more systems are added to a computing environment, the more opportunity there is for failure. If a system is not carefully designed and a single node crashes, the entire system can go down. Distributed systems work without a global clock, requiring careful programming to ensure that processes are properly synchronized to avoid transmission delays that result in errors and data corruption. Under imperfect scalability, it means that doubling the number of nodes in a distributed system does not necessarily double performance. Distributed systems are more complex to design, manage and understand than traditional computing environments. [1].

The "IDify" Distributed database system for digital identification is a system designed to reduce the above-mentioned challenges while ensuring the characteristics of a distributed system such as flexibility, reliability, security, and fault tolerance that the users expect. The main purposes of designing this kind of system are to implement a transaction processing system with extremely low latency, to implement a system that will be resilient to a single point of failure, and to integrate novel database services to provide users with a better and more advanced experience while making sure that their requirements are fulfilled without any erroneous scenarios.

In summary, IDify is a digital identification system that integrates various government services such as personal information management, e-banking, government surveys, authentication, and more. It serves as a comprehensive platform that facilitates seamless access and interaction with these services, providing convenience and efficiency for users. IDify aims to streamline government processes, enhance data accessibility, and deliver a user-friendly experience for individuals engaging with government services.

II. Literature Review

There are various similarly implemented systems to 'IDify' that exist today, such as the Aadhaar system in India, the Nadra platform in Pakistan, and the Turkish electronic ID system. These systems were implemented to provide government services from a centralized platform, aiming to improve resource management and provide reliable services to users.

The Aadhaar system in India is a digital identification platform that provides a unique 12-digit identity number based on biometric and demographic data [2]. It is the world's largest biometric ID system and offers benefits like biometric attendance, passport issuance, and direct benefit transfers.

Nadra in Pakistan is responsible for issuing computerized national identity cards (CNIC) [3]. The CNIC is used for various purposes such as voting, opening bank accounts, obtaining passports, and conducting major financial transactions.

The Turkish eID system replaces paper-based IDs with a two-sided document containing biometric authentication [4]. It is a centralized platform that integrates with various information systems and allows authentication through eID cards.

Saadoon et al. propose an architecture that leverages technologies like HBase, Kudu, and Impala to create a fault-tolerant platform with improved efficiency and reduced latency [5]. They highlight the significance of fault tolerance in big data systems and address common challenges hindering its efficiency.

Shrinivas B. Joshi discusses best practices for tuning Apache Hadoop software and hardware components, emphasizing the importance of configuration tuning for performance benefits. Big data handling systems have wide applications, including customer demand anticipation, predictive modeling, operational efficiency, decision-making, security, and predictive maintenance [6].

Benjamin Vandervalk and Ali Raza compare the performance of Apache Kudu with other streaming data processing systems, focusing on its suitability for real-time analytics [7].

Michael Johnson reviews real-world applications of HBase, highlighting its practical uses in social media analytics, IoT data management, recommendation systems, and fraud detection [8].

Jane Smith evaluates the performance of HBase in large-scale data environments, discussing scalability, performance factors, and optimization techniques [9].

Chang-Shing Lee and Meng Chang Chen provide a comprehensive review of HDFS, discussing its architecture, features, data reliability, fault tolerance, and distributed data processing capabilities [10].

R. Kalpana and M. A. Maluk Mohamed discuss the strengths and limitations of HDFS, including data replication, data locality, access mechanisms, and its suitability for different applications and workloads [11].

T. G. Bhuvaneswari and S. R. Balasundaram compare HDFS with Ceph, evaluating their architecture, data organization, fault tolerance mechanisms, and performance characteristics [12].

III. Methodology

The main objectives of our IDify system are to establish a distributed database system capable of handling millions of requests simultaneously, ensuring fault tolerance to prevent any data loss. We have developed an optimized query engine that can efficiently process queries in a distributed manner. Additionally, implemented microservices-based architecture enables distributed data processing. Furthermore, a centralized operational portal was created to streamline government operations by integrating various government services. By achieving these objectives, we have enhanced the scalability, reliability, and efficiency of the system while providing a seamless user experience.

A. IDify Web Dashboard

The IDify web application is built using the Laravel PHP framework and incorporates JavaScript, AJAX, and jQuery. It features a web dashboard with wireframes for user interfaces and integrates an admin template with Laravel. The dashboard provides authentication, authority management, survey information, and user data services. There are two user roles: admin and authority, each with different access levels upon login.

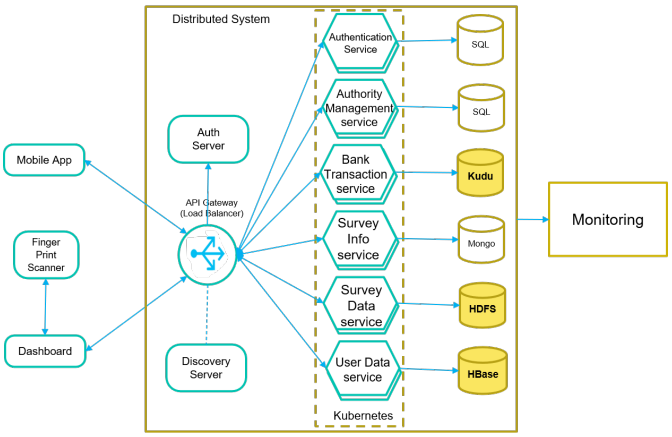

Fig. 1. System Architecture

As an admin, users can perform tasks such as adding and managing authorities, adding users who are all Sri Lankan individuals along with their personal information and other details like driving license information, creating and managing government smart survey forms similar to Google Forms.

The system caters to different authorities such as banks, police departments, and passport offices. Authorities can log in using their credentials and access relevant functionalities based on their role. For example, a bank authority can view customer transactions in the smart wallet.

Microservices were developed to handle these functionalities, communicating with the dashboard through REST API calls. This approach promotes scalability and modularity, enabling independent services to efficiently handle specific tasks. REST APIs facilitate seamless communication and data exchange between the microservices and the dashboard.

B. IDify Mobile Application

The mobile application was built using Flutter and it enables the creation of high-performance, cross-platform apps using a single codebase. The implementation process involved designing UI wireframes and implementing them. API endpoints were also created for essential features like user login, QR scanning, and user profile viewing, allowing communication with distributed databases.

The mobile application supports two user roles: Users and Authorities. Users can access their personal identification details, such as NIC and Driver's License information, as well as other authentication data like Samurdhi, through the app. All user data is stored in the HBase database, eliminating the need for multiple physical cards and providing convenience. Even if a user loses their phone, they can still access their profile using another mobile device.

In the Authorities' role, they can scan users' QR codes to verify their identity and view relevant information. The system retrieves user details from the HBase database through API requests upon scanning the QR code. Authorities, such as policemen, can access users' basic information and driver's license details through the app, preventing the use of fake cards or fraudulent information for authentication.

C. Fingerprint reader

To verify a user's identity, we use their fingerprint. The fingerprint is captured by a device called FPM10A, connected to an Arduino board with an ATmega328P microcontroller. When adding a fingerprint, the user places their finger on

the device twice. The captured data is sent to a dashboard through a Python program and will add to the user's profile. For authentication, the system compares the stored fingerprint with the captured one to confirm the user's identity. This method is especially useful for banks, as it ensures accurate customer authentication using their stored fingerprint data.

D. Hardware Infrastructure

Your system's hardware infrastructure consists of a High-Performance Computer (HPC) with 32 cores, 256GB of memory, and 2TB of storage. To optimize resource allocation, these resources are distributed across a master node and seven worker nodes. The master node has 48GB of RAM and 10 virtual CPUs, while each worker node has 16GB of RAM and 2 virtual CPUs. The HPC runs on the XenServer virtualization platform from Citrix Systems, which allows for efficient creation and management of virtual machines on a host server. This setup enables effective workload distribution and maximizes the performance and efficiency of your system.

E. Distributed System

In the IDify system microservices have been deployed for different kinds of application requirements. Core services handle base functionalities and utility services support them.

Core Microservices

- **Authentication Service** - For registering user roles, identify users by the user ID and update the user roles.
- **Authority Management Service** – To save the authorities, get authority details, and update those details.
- **Survey Info Service** – To create surveys, get survey details, and delete surveys.
- **User Data Service** – For saving users, get user details, update users, and remove users.
- **Bank Transaction Service** – For adding new bank transactions, view all the transactions and view transactions that are related to one user.

Utility Services

- Eureka - Discovery server which acts as the registry of all the available services
- Spring Cloud gateway - API gateway which distributes the load
- Kafka Broker and Zookeeper - For asynchronous communication which maintains a message queue.
- Keycloak with OAuth2 - Provide API authentication and authorization
- Grafana and Prometheus - Monitoring the matrices of the distributed system

F. Deployment

In the IDify system, container creation is managed using Docker and Kubernetes. Kubernetes is set up as a multi-node cluster using Kubeadm, providing scalability for managing containerized applications. Kubeadm simplifies cluster initialization and maintenance tasks, ensuring consistency across platforms. Calico is used as the CNI plugin for networking and network security within the cluster, enabling seamless communication between containers and implementing network policies for enhanced security.

Persistent Volumes (PVs) in Kubernetes are used as independent storage resources that can be dynamically provisioned and utilized by Pods. Due to resource limitations, the system has created two volumes on different nodes within the cluster

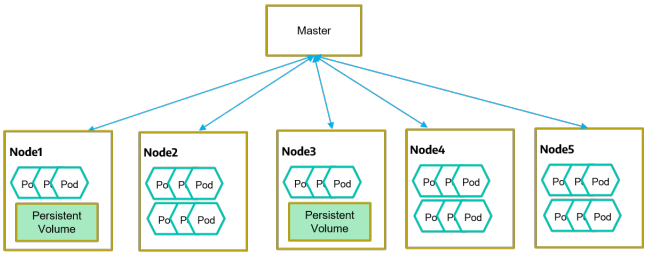

Fig. 2. Kubernetes Architecture

for specific purposes. Persistent Volume Claims (PVCs) are used to request storage by users or Pods and connect applications to the underlying PVs based on their storage needs. For example, the PostgreSQL authentication service is configured with a PVC requesting 100MB of storage from the PV.

G. Distributed Database Technologies

The initial step in designing a distributed database system involves creating the system architecture to meet application requirements. The architecture integrates robust and scalable technologies like Kudu, HBase, and HDFS, implemented within Docker containers.

Kudu is used to implement the critical E-Banking feature. Data in Kudu is organized into tables with their own schema and primary key. Tables are divided into tablets, which are replicated across multiple servers for high availability. One replica serves as the leader for write operations, while any replica can handle read operations. The master node maintains metadata, and the catalog table stores important information.

HBase is used as the central database for user data, authentication, and personal information management. It follows a distributed architecture with an HMaster managing region servers. Multiple regions are combined within a single region server. Docker containers are used for flexibility and scalability.

HDFS is integrated to address data archiving and long-term storage needs. It handles massive amounts of data, providing a distributed file system that scales horizontally. Data blocks are replicated across multiple nodes for reliability and fault tolerance. HDFS supports data compression for optimized storage capacity and cost-effective archiving.

The implemented distributed database system design incorporates Docker containers for Kudu, HBase, HDFS, and Impala. Kudu excels in transaction processing and uses tablets and tablet servers. HBase serves as the central database with robust storage capabilities.

1) Kudu: An open-source distributed columnar storage engine developed by Cloudera. It is designed to provide fast analytics on fast-changing data and is built to be compatible with Apache Hadoop ecosystem tools. Kudu's architecture is based on a combination of different components that work together to provide efficient storage and retrieval of data. [13]

2) HBase: An open-source, distributed, and scalable NoSQL database that provides real-time read and write access to large amounts of structured and semi-structured data.

3) HDFS: A distributed file system designed to store and manage large volumes of data across a cluster of computers. It is a core component of the Apache Hadoop framework and is widely used in big data processing and analytics. [14]

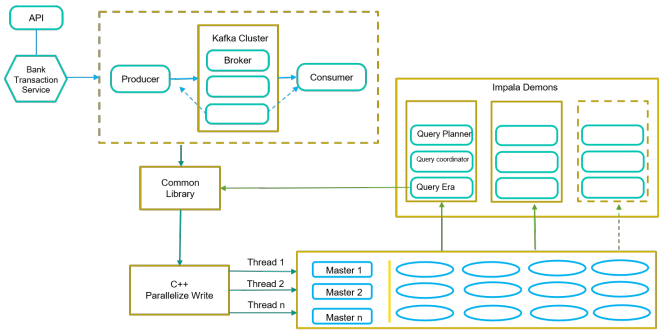

Fig. 3. Kudu Read Write Architecture

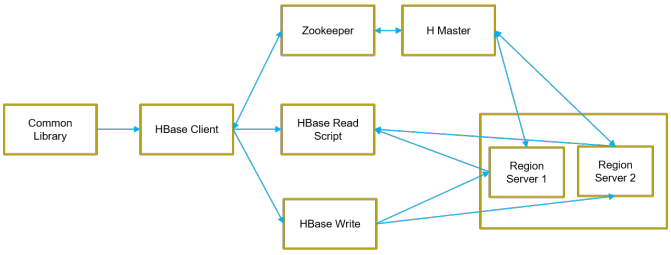

Fig. 4. HBase Architecture

H. Implementing the database read/write Library

The main focus of the project is to implement a Library that integrates various database systems mentioned earlier. The goals include improving the data read/write performance of Kudu, creating specialized APIs for accessing HBase and HDFS file systems, and setting up and configuring Impala.

For writing and reading data from Kudu tables, dedicated APIs and Apache Impala were used respectively. Similarly, a special API was implemented for writing and reading data from HBase tables. Configuration of the HDFS file system was done to support data archiving, and a custom API was integrated for easy access. The main objective was to enhance the performance of data read and write operations in Kudu. [15]

To improve Kudu's write performance, parallelism was implemented using the threading system of processors. This technique utilized multiple threads to significantly enhance data write operations. Impala was carefully configured by fine-tuning and optimizing internal parameters to achieve optimal performance based on the specific application requirements.

The IDify project focuses on integrating diverse database systems, optimizing data processing in Kudu through parallelization, and configuring Impala for optimal performance. The aim is to improve the overall efficiency and effectiveness of data management in the system.

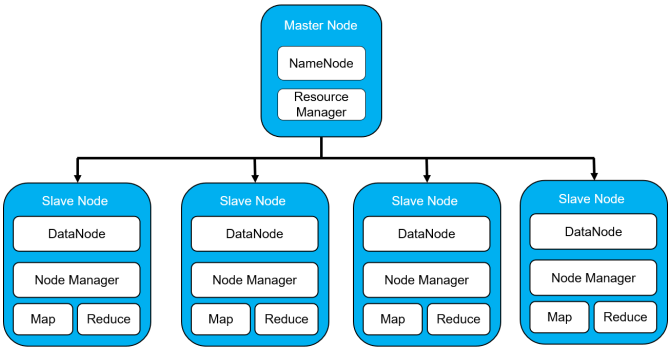

Fig. 5. HDFS Architecture

1) Kudu Write Operation:

a) Creating Kudu C++ Connection: The Kudu C++ client library was set up in the project, ensuring proper linking and installation of dependencies. It provided classes and functions for interacting with Kudu, and the library was connected to the Kudu cluster through creating a KuduClient object and master addresses. The connection was established to the desired Kudu table, and a KuduSession object was created to group operations and maintain consistency. The schema of the Kudu table was obtained using the schema() function, and the KuduInsert object was created to insert a new row. The insert operation was applied to the Kudu table using the Apply method of the KuduSession object, ensuring consistent execution. [16]

Errors or exceptions were handled gracefully using the status() method of the KuduSession object. These steps allowed for effective interaction with Kudu tables using the C++ library.

b) Parallelizing the Implemented C++ Script: Kudu is well-suited for scenarios requiring low-latency random access and high-throughput analytics on large datasets. It is suitable for storing time-series data, machine-generated data, and analytical data. In the IDify system, Kudu is used to store bank transaction data, which falls under the category of time-series data.

To parallelize the write performance of Kudu using C++, the data is divided into smaller chunks and assigned to separate worker threads. These threads execute write operations concurrently, leveraging parallelism to enhance performance. Synchronization mechanisms are used to ensure thread safety and data integrity. By utilizing multiple threads, the workload is distributed effectively, leading to improved performance when writing data to Kudu tables.

The main purpose of using threads in this code is to parallelize data processing and insertion tasks, which can boost performance by utilizing multiple CPU cores and reducing execution time. The implemented script executes a loop to create and start the threads. Each thread handles a specific chunk of lines, while the session and table objects are shared between all threads.

After creating the threads, the main thread waits for each thread to finish processing by calling the join() function on each thread object. By dividing the work into smaller chunks and processing them concurrently with multiple threads, the code takes advantage of parallelism, potentially accelerating the data insertion process.

Using threads to achieve parallelism offers the potential for improved performance in the data insertion process. By leveraging multiple threads, the workload can be divided and processed simultaneously, leading to faster execution and enhanced overall efficiency.

2) Configuring Impala for Kudu Read Operation:

a) Apache Impala: Apache Impala is an open-source query engine designed for high-performance analytics on big data. It operates within the Apache Hadoop ecosystem and enables interactive and real-time querying of large-scale datasets stored in HDFS and HBase.

Impala offers a SQL-like interface, allowing users to query and analyze data in Hadoop without the need for data movement or transformation. It supports various SQL operations such as joins, aggregations, filtering, and complex analytics functions. By executing queries directly on the data nodes, Impala reduces latency by eliminating data movement, resulting in near real-time response times for queries. [17]

b) Impala Query Handling: IDify's Transaction Processing system sends queries to Apache Impala, which is received by the Impala Coordinator, a crucial component of the Impala daemon. The Coordinator parses the SQL statement, performs syntactic and semantic analysis, and generates an execution plan for efficient query execution. To optimize query execution, the Impala Coordinator requires access to metadata information from Apache Hive, which stores schema and other relevant information about bank transaction data. After the parsing, planning, and metadata retrieval stages, the Impala Coordinator distributes query tasks across Impala daemons in the cluster. Each task represents a portion of the query that can be executed independently on a specific subset of the data.

In IDify's database configurations, three Impala daemons independently operate on the designated KUDU database. They receive query tasks from the Coordinator, which computes and returns intermediate results. These results are aggregated or processed to generate the final set. After query execution, the Impala Coordinator sends the result set to the client for analysis or visualization. Impala's architecture, with parallel processing and in-memory computing, ensures high-performance, low-latency queries ideal for real-time analytics on big data. [18]

IV. Experiments and Results

For the testing dataset, we have generated one million data rows containing personal user information, such as name, email, and telephone, along with dummy fingerprint data, etc. The comparison of data read and write performance among KUDU, HBase, and HDFS involved analyzing latency values against different dataset partitions. This analysis provided insights into the scalability and efficiency of each technology for read and write operations. The findings help guide the selection of an appropriate database technology based on workload requirements, enabling efficient data processing systems.

TABLE 1 : COMPARISON OF EXECUTION TIME FOR DIFFERENT CONFIGURATIONS IN IMPALA

Test	Configuration	Default Value	Set Value	Latency
Query Option Configurations				
Test 01	With default configurations			7s
Test 02	MT_DOP	0	12	5s
Test 03	MT_DOP	0	16	3s
Test 04	MT_DOP	0	24	9s
Test 05 with Test 01 Configuration	MT_DOP	0	0	6s
	NUM_SCANNER_THREADS	0	10	
Test 06 with Test 02 and Test 05 Configurations	MT_DOP	0	12	3s
	NUM_SCANNER_THREADS	0	10	
Test 07 with Test 01 Configuration	MT_DOP	0	0	5s
	NUM_SCANNER_THREADS	0	20	
Test 08 with Test 01 Configuration	MT_DOP	0	0	8s
	NUM_SCANNER_THREADS	0	5	

Above table shows the test results after the discovered configurations were utilized in combinations. These tests were performed on the 1 million dataset which was fed into KUDU tables. Basically a 'SELECT *' was executed and tested in each of these following test cases.

MT_DOP is a query option which sets the degree of intra-node parallelism used for certain operations that can benefit from multithreaded execution. I can be can specified values higher than zero to find the ideal balance of response time, memory usage, and CPU usage during statement processing

NUM_SCANNER_THREADS is a query option which sets the maximum number of scanner threads (on each node) used for each query. By default, Impala uses as many cores as are available (one thread per core). You might lower this value if queries are using excessive resources on a busy cluster. Impala imposes a maximum value automatically, so a high value has no practical effect

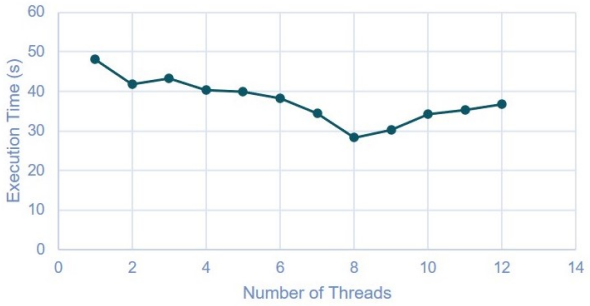

Fig. 6. KUDU 1 million dataset ingestion Comparison on variable threads

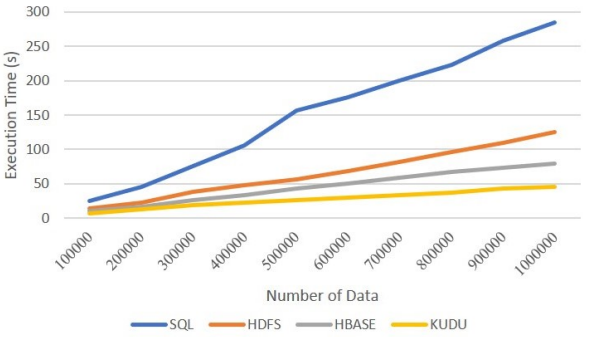

Fig. 7. SQL vs HDFS vs HBase vs KUDU data Write performance comparison on different datasets

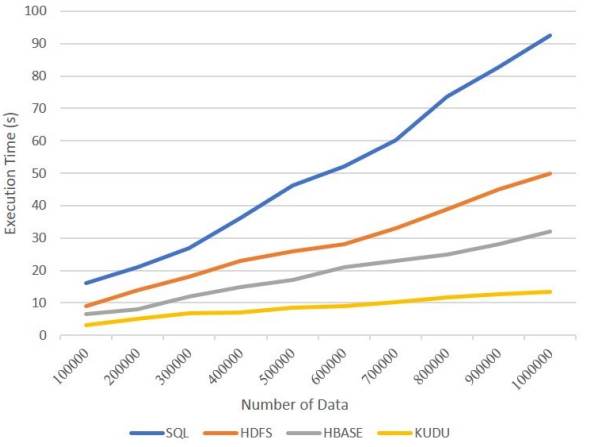

Fig. 8. SQL vs HDFS vs HBase vs KUDU data read performance comparison on different datasets

V. Challenges

One of the main challenges faced during the project was the lack of community support for Hadoop, resulting in difficulties finding resources and support. This required exploring alternative sources of information and platforms. Another challenge was the need for a large-scale hardware infrastructure, which was addressed by leveraging high-performance machines or cloud services. Accessing the HPC machine remotely introduced dependency on stable network connections and software stability, requiring physical presence for issue resolution. The configuration of different versions of HDFS, HBase, and Kudu, along with their compatibility and integration, posed challenges. Version mismatches and incompatible read-write libraries complicated system stability. Despite these challenges, alternative resources and careful management allowed successful implementation of the distributed system.

VI. Future Works

Future research in distributed database systems will focus on performance optimization, including reducing execution time and improving storage techniques. This involves exploring data structures, compression algorithms, and caching strategies to enhance data access and minimize network overhead. Integration with machine learning applications will be a key direction, enabling efficient data ingestion, distributed training, and real-time predictions. Distributed databases will also need to handle diverse data formats beyond structured models, such as JSON and Parquet, to accommodate modern data sources. Optimizing deployment in cloud environments and exploring auto-scaling mechanisms will further enhance scalability and cost-efficiency.

VII. Conclusion

IDIFY employs a diverse range of database technologies tailored to specific requirements. For storing user data, IDIFY relies on HBase due to its scalability and high performance, making it ideal for managing individual user details. Kudu is the preferred choice for storing bank transaction data, offering efficient storage and processing capabilities that are well-suited for daily data migration tasks. Meanwhile, HDFS is used for managing large-scale survey data, providing reliable storage and parallel processing capabilities.

Impala acts as the primary query engine, enabling fast and interactive data analysis across Kudu, HDFS, and HBase. Additionally, the Hive Metastore serves as the central repository for metadata. To optimize parallel data writes to Kudu through Impala, a custom C++ library has been developed, leveraging 8 threads. However, the optimal number of threads may vary depending on the dataset.

To ensure robustness and fault tolerance at the architectural level, IDIFY relies on Kubernetes and robust failure-handling mechanisms. These measures guarantee system reliability, even in the face of unexpected failures, ensuring that IDIFY's data infrastructure remains stable and responsive.

The proposed IDify system excels over traditional approaches with its distributed architecture, ensuring scalability for a growing user base. Leveraging modern database technologies, parallel processing, and fault-tolerant mechanisms enhances system performance, making IDify ideal for the dynamic needs of digital identification services, delivering a more robust and responsive solution compared to traditional systems..

Acknowledgment

We extend our heartfelt gratitude to Dr. Nathali De Silva, our supervisor, and Ms. Dilani Ranaweera, our co-supervisor, for their unwavering support, guidance, and encouragement throughout our project. Their invaluable advice has shaped our work and propelled us toward success.

We also express our appreciation to Dr. Randima Dinalankara, the former head of the department, Dr. Udaya Wijenayaka head of the department as well as the lecturers, instructors, academic staff, and non-academic staff for their guidance and provision of necessary resources. Their contributions have been instrumental in our growth and development as undergraduates.

References

[1] M. T. Özsu and P. Valduriez, Principles of Distributed Database Systems. Cham: Springer International Publishing, 2020

[2] "Aadhaar." [Online]. Available: https://covidinfo.rajasthan.gov.in/

[3] admin, "NADRA Pakistan – National Database and Registration Authority Official Website," Oct. 2015. [Online]. Available: https://www.nadra.gov.pk/

[4] M. Mutlugün and O. Adalier, "Turkish national electronic identity card," in Proceedings of the 2nd international conference on Security of information and networks - SIN '09. North Cyprus, Turkey: ACM Press, 2009, p. 14.

[5] M. Saadoon, S. H. Ab. Hamid, H. Sofian, H. H. M. Altarturi, Z. H. Azizul, and N. Nasuha, "Fault tolerance in big data storage and processing systems: A review on challenges and solutions," Ain Shams Engineering Journal, vol. 13, no. 2, p. 101538, Mar. 2022.

[6] L. Zhu, X. Cai, and Y. Le, "Research on Performance Optimization for Power Big Data Storage based on HBase," Journal of Physics: Conference Series, vol. 2033, no. 1, p. 012181, Sep. 2021.

[7] Vandervalk, B. and Raza Butt, A. (no date) '"Exploring the Performance of Apache Kudu for Stream Processing Workloads"', IEEE International Conference on Big Data (Big Data), 2018 [Preprint].

[8] M. Johnson, "HBase Use Cases: A Review of Real-World Applications," 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), 2020. doi:10.1109/dsaa49011.2020.00121

[9] J. Smith, "Performance Analysis of HBase in Large-Scale Data Environments; ," Journal of Big Data and Analytics, 2019.

[10] C. Shing, HDFS: A Distributed File System for Big Data, 2018.

[11] Kalpana and M. A. Maluk Mohamed, "Hadoop Distributed File System (HDFS) ," March 29-30, 2015 Singapore, 2015. doi:10.17758/ur.u0315253

[12] T. G. Bhuvaneswari and S. R. Balasundaram, "A Comparative Study of HDFS and Ceph Distributed File Systems," 2018 IEEE High Performance extreme Computing Conference (HPEC), 2018.

[13] "F. lefebvre-naré, s. lemire, and g. j. petersson, "what is big data?," in cyber society, bigdata, and evaluation, new brunswick: Transaction publishers, [2017] — series: Routledge,2017, pp. 19–34."

[14] "Big Data Analysis Architecture," Economic Alternatives, vol. 27, no. 2, Jun. 2021.

[15] T. Lipcon, M. Percy, D. Alves, D. Burkert, J.-D. Cryans, A. Dembo, S. Rus, D. Wang, M. Bertozzi, C. P. McCabe, and A. Wang, "Kudu: Storage for Fast Analytics on Fast Data."

[16] M. U. Hassan, I. Yaqoob, S. Zulfiqar, and I. A. Hameed, "A Comprehensive Study of HBase Storage Architecture—A Systematic Literature Review," Symmetry, vol. 13, no. 1, p. 109, Jan. 2021.

[17] K. J. Merceedi and N. A. Sabry, "A Comprehensive Survey for Hadoop Distributed File System," Asian Journal of Research in Computer Science, pp. 46–57, Aug. 2021.

[18] R. W. A. Fazul, P. V. Cardoso, and P. P. Barcelos, "Improving Data Availability in HDFSthrough Replica Balancing," in 2019 9th Latin-American Symposium on Dependable Computing (LADC). Natal, Brazil: IEEE, Nov. 2019, pp. 1–6.