IJARASEM

ISSN: 2395-7852

International Journal of Advanced Research in Arts, Science, Engineering & Management

Volume 12, Issue 1, January- February 2025

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.583

+91 9940572462 +91 9940572462 ijarasem@gmail.com www.ijarasem.com

International Journal of Advanced Research in Arts, Science, Engineering & Management (IJARASEM)

| ISSN: 2395-7852 | www.ijarasem.com | Impact Factor: 7.583 | Bimonthly, Peer Reviewed & Referred Journal|

| Volume 12, Issue 1, January- February 2025 |

Creating a File System Architecture to Address Health Large Data Archiving and Storage Issues with a Distributed File System

Ligaya O. De Castro, Jerry I. Teleron

Department of Graduates Studies, Surigao del Norte State University, Surigao City, Philippines

ABSTRACT: The use of computers, smartphones, tablets, Internet of Things (IoT) devices, and other digital sources has increased recently as internet usage has grown widely. This digital world has also tended to grow in an unpredictable fashion in the health sector, which has roughly 10% of the world's data and is still growing faster than the other sectors thanks to new generation digital medical equipment. This development has significantly increased the volume of data generated that is unresolvable using traditional techniques. An effective model for storing medical photographs using a distributed file system structure has been created in this work. A serverless, scalable, reliable, and available solution structure has been created with this effort, particularly for the storage of massive volumes of data in the medical industry. Additionally, by using synchronously encrypted file contents, user credentials, and static Internet protocol (IP), the system's security level is quite high. The system's easy scalability and fast performance are among its most crucial qualities. This makes the system more resilient than others that employ name node architecture and allows it to function with less hardware components. Compared to other systems that use name node architecture, this one can function with less hardware components and be more reliable. Based on the test findings, the designed system outperforms a Not Only Structured Query Language (NoSQL) system by 97%, a relational database management system (RDBMS) by 80%, and an operating system with an OS by 74%.

KEYWORDS: health data, medical imaging, distributed file systems, and big data

I. INTRODUCTION

Information technology has advanced globally in recent years, and the amount of data generated in all disciplines has expanded tremendously due to internet usage. The quantity of people using the internet was In 1995, 16 million. 2000 saw 304 million, 2005 saw 888 million, 2010 saw 1.996 billion, 2015 saw 3.270 billion, and 2017 saw 3.885 billion [1–3]. Globally, 2.5 exabytes (EB) of data are created every day. Additionally, since 2015, 90% of the data generated worldwide has been produced. The data produced spans a wide range of industries, including the energy, aviation, meteorological, IoT, and health sectors. Similarly, the amount of data generated by social media has increased significantly. In 2014, Google processed hundreds of petabytes (PB) of data every day, in addition to Facebook.com storing 600 TB of data daily [4,5]. The health sector has also seen a notable growth in data generation, which has been sparked by the growing adoption of digital medical imaging peripherals. Additionally, the amount of data generated in the health industry has increased to the point that it is difficult to handle using conventional data management gear and software. By maintaining patient records, developing medical imaging to aid physicians in diagnosis, preserving survey results, and converting numerous gadgets into digital format, the healthcare industry has amassed a large number of data. Data from a variety of sources, including patient records, lab findings, X-ray machines, computed tomography (CT) scans, and magnetic resonance imaging (MRI), can be both structured and unstructured. The number of patients to be served is expected to grow exponentially due to the apparent continual increase in the world's population and average longevity. The volume of data gathered rises sharply in tandem with the number of patients. Additionally, comprehensive digital healthcare devices facilitate the addition of higher-density graphical outputs to the expanding data set. The U.S. healthcare sector's data quantity in 2011. It seemed to have reached 153 EB in 2013. This figure is predicted to rise to 2.3 ZB in 2020. For instance, the Electronic Medical Record (EMR) grew by 31% between 2001 and 2005 and by over 50% between 2005 and 2008 [6, 7]. Although the data quantities for neuroimaging operations increased to about 200 GB yearly between 1985 and 1989, they really increased to 5 PB annually between 2010 and 2014, indicating an increase in data in the health sector [8].

In this way, the growing amount of data generated globally in all disciplines has led to the emergence of new issues. The data storage and analysis now provide significant hurdles. It is now more expensive to store data than to collect it [9]. The production, storage, and manipulation of data has therefore expanded significantly, leading to the emergence of "big data" and data science/knowledge [10]. Variety, velocity, and volume are the three ideas that make up "big data." It

might be particularly challenging to find a suitable method to address these problems when it comes to medical information.

Big data problems in healthcare and the objectives of the study according to the previous arguments are listed as follows:
1.Growing number of patients: both the average lifespan and the world's population appear to be growing. For instance, since 2012, the number of doctor visits in Turkey has increased by almost 4% year [11]. Additionally, the number of visits to a doctor per capita in health care facilities was 8.6 in 2016 compared to 8.2 in 2012. More data must be controlled because as the number of patients rises, so does the volume of data that is gathered.
2.The vast amount of data that needs to be saved is indicated by the high-resolution graphical outputs produced by comprehensive digital healthcare devices.
3.Expert staff requirements: qualified information technology specialists must be employed to deploy, administer, and store big data solutions in order to manage big data in institutions employing Hadoop, Spark, Kubernetes, Elasticsearch, etc.[12].
4.Small file size issue: existing healthcare solutions, including Hadoop-based solutions, have 64 MB block sizes, as explained in the next section. This causes performance flaws and needless storage utilization, known as "internal fragmentation," which is challenging to fix.
5.Hospital Information Management Systems (HIMS) are all-inclusive software and associated technologies that facilitate improved patient tracking and care by assisting healthcare providers in creating, storing, retrieving, and exchanging patient data more effectively. Important non-functional characteristics of HIMS are (a) availability, (b) performance, (c) scalability, and (d) resilience. These characteristics mostly rely on the data management architecture that has been built, which comprises installed software tools and configured hardware. HIMS is solely in charge of resolving big data issues. HIMS is far more than just a conventional application program or an IT project. In order to accomplish the goals of the healthcare providers, third-party software tools are required.

The goal of this project is to acquire a middle-layer software platform that will aid in filling these gaps in healthcare. Stated differently, a server-cluster platform has been put in place to store and retrieve health digital image data. It serves as a link between HIMS and different network hardware resources. This study has five main objectives:
1. To construct a distributed data layer between the server-cluster platform and HIMS in order to address the rising data problem.
2. This solution lowers operating expenses by eliminating the need to hire IT specialists for the installation and deployment of well-known big data technologies.
3. To put into practice a novel distributed file system architecture in order to address non-functional characteristics that are vital to HIMS, such as performance, security, and scalability.
4. To demonstrate and substantiate the possibility of many effective big data solutions.
5. In particular, to effectively address these deficiencies for our university's HIMS.

The first component of this paper outlines common data processing techniques. The literature on the topic is covered in the second section, and the materials and methods portion, which outlines the adopted strategy, is covered in the tird. The evaluation that results from our labor is concluded in the final part.

II. OBJECTIVE OF THE STUDY

The objective of the study focused on creating a file system architecture to address health large data archiving and storage issues with distributed file systems is to design a scalable, efficient, and secure method for managing vast amounts of health-related data across distributed systems. Specifically, the study would aim to achieve several key goals:
1. Scalability: Develop a file system that can efficiently handle the increasing volume of health data, such as patient records, medical imaging, and genomic data, which continues to grow rapidly with advancements in healthcare technologies.
2. Fault Tolerance and Redundancy: Ensure that the system can maintain data availability and integrity even in the event of hardware failures, network disruptions, or other potential risks. This would involve implementing redundancy mechanisms such as replication across multiple nodes in the distributed system.
3. Performance Optimization: Optimize data retrieval and storage processes to ensure that large datasets can be quickly accessed and processed, especially in critical healthcare environments where time-sensitive access to data is crucial.
4. Security and Privacy: Address healthcare-specific regulatory requirements like HIPAA (Health Insurance Portability and Accountability Act) by incorporating strong encryption, authentication, and authorization mechanisms to protect sensitive medical data from unauthorized access.

5. Cost Efficiency: Develop a cost-effective solution by leveraging distributed resources, optimizing storage usage, and reducing the overhead involved in managing large health data volumes across multiple geographic locations.
6. Data Integrity and Long-Term Preservation: Design the file system to ensure the integrity and durability of health data, addressing challenges related to long-term data storage and ensuring that archived data remains accessible and uncorrupted over extended periods of time.
7. Interoperability and Standardization: Ensure that the proposed file system can integrate with existing healthcare infrastructure, such as electronic health records (EHR) systems, medical devices, and cloud platforms, while adhering to established standards for healthcare data formats (e.g., DICOM, HL7).

Relational database management systems (RDBMS) and NoSQL database systems are two prevalent application architectures found in database systems. The most well-known and popular systems for this purpose are RDBMSs, which are structured data storage systems. The right kind and format of data must be used for processing. A single database can support numerous users and applications in these systems. These systems have many limitations, including atomicity, consistency, isolation, and durability, and the data structure needs to be specified beforehand because they are based on vertical growth functionality. Today, there is a growing skepticism regarding the stringent regulations that render these systems essential. However, the initial installation expenses are considerable because of the technology and software that are being employed. One of the main reasons they aren't used in big data issue solving is that the horizontal scalability feature will be very unsatisfying and challenging to maintain, especially as the volume of data increases. Furthermore, compared to file systems, these systems are more complicated.

It, above all, is not appropriate for huge data. The inability of RDBMSs to handle large amounts of data has led to the emergence of NoSQL database systems as a substitute. These systems are primarily designed to store the growing amounts of internet data and to respond to the demands of high-traffic systems using semi-structured or unstructured forms. According to RDBMSs, NoSQL databases are systems that offer high accessibility and facilitate horizontal data scaling [13]. Performances in reading and writing can be more acceptable than RDBMS. Their ability to spread horizontally is one of their most crucial characteristics. Big data can be processed by a cluster of thousands of servers. Their flexible structures make them simple to manage and program. Data processing rates have increased as a result of these systems' requirement to perform grid computing in clusters, which are made up of numerous machines connected to a network. In addition, its data security measures are not as sophisticated as those of RDBMS. Professional technical assistance and documentation are absent in several NoSQL initiatives. Additionally, NoSQL database systems lack the idea of transactions and are not appropriate for use in banking and financial systems due to the possibility of data loss [14].

The second method, referred to in the literature as "file servers," uses the fundamental file management features of operating systems (OS). The underlying operating system file structure contains files and folders where medical image data is kept in this system. The file system used by operating systems is hierarchical. The files in this structure are arranged in a "directory," which is a tree structure. HIMS determines how file servers store files based on patient information, policlinic name, image kind, and file creation date. Through the use of system calls, which give storage devices a low-level interface with the assistance of the operating system, HIMS carries out read and write operations. The benefits of using file servers are their acceptable file operation performance, ease of implementation, and simplicity of delivery. Because the operating system has been designed with file management in mind, writing, deleting, reading, and searching files on it is a reasonably quick procedure. Operating systems, in particular, are more flexible and perform better than RDBMS and NoSQL systems. However, the OS's lack of horizontal scalability prevented it from utilizing these benefits to be a good big data solution model. Serving system calls to other apps is OS file management's primary responsibility. Therefore, the OS is not the solution by itself, but rather a component of it. Data cannot be scaled to the size of the data, backup and file transfers cannot be done reliably, and data storage is not as secure as other ways. It appears that huge data issues cannot be resolved by the operating system alone.

Distributed File System (DFS), sometimes known as "distributed file systems," is the third technique. These are the most recent systems that assume that machines in different places are part of a same framework and offer the best solution to the big data challenge. Big data analytics and storage are the main uses for Hadoop DFS and MapReduce. The literature also discusses the usage and criticism of hybrid systems, such as Hadoop and NoSQL. The use of the Hadoop ecosystem in the healthcare industry does have certain disadvantages, though. The first is the little files issue, which means Hadoop is unable to effectively store and handle these kinds of files [15]. An example of this would be a 1 GB file that contains 16 64 MB Hadoop pieces.

A name node contains 2.4 KB of space. Nevertheless, 100,000 100 KB files take up 1 GB of data node capacity and 1.5 MB of name node space. This implies that processing small files requires more MapReduce operations. Given that the typical size of medical picture files in the healthcare industry is 10 MB, a new DFS system is required to support

systems with a lot of little files. This paper suggests a novel approach to this problem by using a node structure without a name and a short block size. This study was conducted for 26 healthcare organizations from Europe and the United States that employ Hadoop/MapReduce [16]. However, this study also notes that more thorough analytical methods, such as deep learning algorithms, are needed to meet the demands of an increasing volume of unstructured data. Liu and Park provided a thorough examination and discussion of Hadoop/MapReduce and Storm frameworks for big data in healthcare. It claimed that a performance gap prevents Hadoop/MapReduce from being utilized in real-time systems. As a result, they have put forth the BDeHS architecture, a unique health service paradigm with three main advantages. On the other hand, MapReduce claims that Spark and Storm are better suited for real-time data analytics of big datasets [17]. One study offered comprehensive details on the architectural layout of the "MedCloud" personal health record system, which was built on top of Hadoop's ecosystem [18]. Another study by Erguzen and Erdal examined big data in healthcare. To store regions of interest (ROIs) from MRIs, a new file format and achievement system were created. To put it another way, they took the ROI parts of the image—which held important patient information—and threw away the rest, or non-ROIs. The ROIs were then saved in the newly created file structure, with a success rate of roughly 30%. However, the goal of this work was not to efficiently store vast data on DFS, but rather to reduce the image sizes [7]. The other study by Raghupathi and Raghupathi revealed that the Hadoop ecosystem has major disadvantages for medium- or large-scale healthcare providers: (a) it necessitates a high level of programming expertise for MapReduce data analytics tasks; and (b) it is nearly impossible to install, configure, and manage the Hadoop ecosystem entirely, making it impractical for medium- or large-scale healthcare providers [12].

One of the enterprise-scaled open source solutions available today is Hadoop, which enables the storage of large amounts of data with thousands of data nodes and the use of MapReduce for data analysis. Hadoop does have three drawbacks, though. The first is that handling many little files is made difficult by Hadoop's 64 MB block size by default [15]. Internal fragmentation is the gap created when a file smaller than 64 MB is placed in a 64 MB Hadoop block. The block size on our system is 10 MB, which indicates less internal fragmentation because it was built based on the typical MRI file size. Second, when the system must function in a real-time setting, performance issues become a major concern. The third problem with Hadoop is that proper system construction, operation, and maintenance necessitate expert assistance. These disadvantages are the main reasons we came up with this possible remedy. In order to store and manage healthy huge data, a unique distributed file system has been created. For applications that use the write once read many (WORM) model—which has several applications in the healthcare industry and electronic record management systems, for example—the created method has proven to be highly effective.

III. LITERATURE REVIEW

Here are some recent works from 2023 and 2024 that you can consider including in your literature survey on developing a file system structure to solve healthy big data storage and archiving problems using a distributed file system:

1. "15 years of Big Data: a systematic literature review" (2024)

The current state of the art in Big Data research during the previous 15 years is summed up in this thorough literature review. It offers information on key application areas, noteworthy difficulties, and new research directions.

2. "Comparative Analysis of Object-Based Big Data Storage Systems on Architectures and Services: A Recent Survey" (2024)

The architectures, services, and prospects for the future of object-based big data storage systems are examined in this review study. It aids in differentiating the fundamental traits of object-based storage systems and how they are implemented.

3. "Developing a File System Structure to Solve Healthy Big Data Storage and Archiving Problems Using a Distributed File System" (2023)

An effective methodology for storing medical photographs utilizing a distributed file system structure is presented in this research. It emphasizes the system's robustness, easy scalability, and excellent performance.

Big data is information that cannot be managed and stored on a single computer. These days, computers that are part of a network and connected to a distributed file system are utilized to administer it. DFSs are divided up into node-based clusters. The key characteristics of big data are performance, data security, scalability, availability, robustness, and reliability. DFS and network infrastructure are used to overcome big data management issues. The 1970s saw the start of DFS-related works [19]. The Roe File System, one of the earliest experiments, was created for network transparency, secure file authorization, easy setup, and replica consistency [20]. Another DFS with network transparency, great performance, and high reliability is LOCUS, which was created in 1981 [21]. In 1984, Sun Microsystems began developing the Network File System (NFS). On UNIX, this system is the most frequently used DFS. Communication takes place using remote procedure calls (RPCs) [22]. It is intended to make the Unix file system capable of operating

as a "distributed" system. One layer is the virtual file system. As a result, clients can run many file systems with ease, and the NFS has a high fault tolerance. Information about file status is stored, and the client promptly notifies the server of any errors. While the entire system is replicated via NFS, file replication is not [23]. NFS only allows file system sharing; printers and modems cannot be shared. Both files and directory units can be objects to be shared. Every application does not have to be installed on a local disk with NFS; they can be shared via the server. Despite this, a single machine can operate as both a client and a server. Consequently, NFS lowers the cost of data storage.

OpenAFS [24], CODA (1992), and the Andrew file system (AFS-1983) are all open sources distributed file systems. The cluster size of these systems is greater and more scalable. They can also cache the entire file and lessen server demand. In order to make CODA more accessible, it replicates on several servers. OpenAFS and CODA support MacOS and Microsoft Windows, while AFS only supports Unix. Every client in these systems has the same namespace allocated for them. Nevertheless, read-one/write-all (ROWA) architecture is utilized for replication, which has restricted capabilities [25, 26].

In 1997, Frangipani, a new distributed file system with two levels, was created. The virtual drives make up the bottom tier. They offer storage services. In addition to being automatically handled, it may be scaled. The Frangipani file system is used by a number of machines on the top layer. On the shared virtual drive, these computers operate in a distributed manner. The same collection of files can be accessed consistently and collectively through the Frangipani file system. Higher performing hardware components and additional storage space are required as the system's data usage increases. Due to its availability, the system still functions even if one of its parts fails. There is less need for human management as the system expands because the additional components do not complicate management [27].

Distributed over a network, FARSITE (2002) is a serverless file system. On a network of physically faulty computers, it operates in a distributed manner. The system is a distributed file system without a server. There is no need for centralized administration. As a result, unlike a server system, there are no personnel expenses. The file input/output workload of a desktop computer in a large organization or university is supported by FARSITE. It offers scalability through namespace delegation, availability and accessibility through replication, authentication through encryption, and decent speed through client caching. Utilizing the advantages of Byzantine fault-tolerance is one of FARSITE's primary design objectives [28].

The CEPH file system, which was described in 2006, sits on top of other systems that do object storage. Data and metadata management are separated by this layer. For unstable object storage devices (OSDs), the random data distribution function (CRUSH) is used to achieve this. The file allocation table is replaced by this function. CEPH transfers distributed data replication, error detection, and recovery activities to local file system-based object storage devices. As a result, system performance is improved. The management of a distributed set of metadata is quite effective. All filing operations are controlled by the Reliable Autonomic Distributed Object Store (RADOS) layer. To test the performance of CEPH, which may operate with varying disc sizes, measurements were made under a range of workloads. I/O performance is quite high as a result. Its scalable metadata management has been demonstrated. It can process 250,000 meta transactions per second because to the measurements. A scalable, dependable, and high-performance distributed file system has been created with CEPH [29].

Hadoop, which includes the MapReduce parallel computing engine and the Hadoop distributed file system (HDFS), was created in 2007. Very massive datasets can be analyzed and transformed using the Hadoop platform. HDFS uses clusters on regular servers to distribute large data. backs up the blocks on the servers by replicating them in order to guarantee data security [30]. Big data is processed and managed using Hadoop/MapReduce. The data is dispersed throughout the cluster and made available for processing by the "map" function. The data will be merged thanks to the "reduce" function. Hadoop can readily accommodate PBs of data and is scalable [31]. Many large companies use Hadoop today. In academic and industrial domains, it is favored. Hadoop is widely used by businesses like LinkedIn, eBay, AOL, Alibaba, Yahoo, Facebook, Adobe, and IBM [32].

CalvinFS, a file system with replica property and scalability, was introduced in 2015 and uses an extremely effective database for metadata management. It accomplishes this by horizontally dividing the metadata over several nodes. Distributed file actions are required to modify the metadata item. Standard file systems are also supported by this system. It has been demonstrated that scaling to billions of files is possible with this file system architecture. It can simultaneously process millions of readings per second and hundreds of thousands of updates while minimizing reading latency [33].

Mohammed S. Al-Kahtani and Lutbul Karim introduced a framework for scalable distributed systems in 2016 [34]. Scaling is done by the system on the central server. As more data is gathered, the server in the suggested structure shifts

the data processing workload to additional machines. In other words, when the amount of data increases, the system functions in a distributed manner. Similar frameworks include the IdeaGraph algorithm [35], parallel two-pass MDL (PTP-MDL) [40], item-based collaborative filtering algorithm [39], recommendation algorithm [40], convex optimization [41], locality-aware scheduling algorithm [37], nearest neighbor algorithm [38], and probabilistic latent semantic analysis (PLSA) [36].

An effective electronic health record storage table was created by Yang Jin et al. Using MapReduce, the system also does analysis and generates comprehensive statistics values. It is built using HBase, a distributed-column database that utilizes the MapReduce framework of the Hadoop distributed file system. The model contains two name nodes and is inexpensive. To improve performance, load balancing is also permitted by HMaster and HRegionServer. But it has been observed that the system ought to endeavor to create HDFS data block replication techniques [41].

There are now two primary categories in which distributed file systems can be thought of or studied:
- Massive data storage: To save massive data (data save), use the appropriate file system and cluster structure.
- Big data analytics: The condensed and reliable examination of data gathered from nodes using grid computing technologies (data mining).

IV. METHODS

This section discusses the state-of-the-art technologies and the related algorithms used in this study.

4.1. TCP/IP Protocol

Strict guidelines for message exchange between communicating entities via a local area network or wide area network are specified by protocols. Every Internet-connected computer or mobile device has a single, distinct IP address that cannot be shared by other Internet-connected devices. By employing packets or datagrams that contain source and destination device IP addresses as well as other relevant information, the Internet Protocol (IP) connects endpoints—which are made up of an IP address and a port number. The User Datagram Protocol (UDP), Lightweight User Datagram Protocol (UDP-Lite), Transmission Control Protocol (TCP), Datagram Congestion Control Protocol (DCCP), and Stream Control Transport Protocol (SCTP) are among the transport layer protocols required by this IP. The most used protocols for internet connections are TCP and UDP. Among the many benefits that TCP offers over UDP are (a) streaming, (b) ordered packet transfer, and (c) reliability. TCP/IP is therefore utilized in this project.

4.2. Sockets

A TCP socket is a process endpoint for client/server communication that is identified by an IP address and port number. As a result, a TCP socket is only one endpoint of a two-way communication relationship between two processes; it is not a connection. Client and server sockets are the two varieties that have the ability to send and receive. Therefore, the way the connection is made is where they differ from one another. While the server socket continuously listens to the designated port for client requests, the client sockets establish the connection. A socket library for the.NET framework was utilized in this project.

4.3. Windows Services

The Microsoft Windows operating system has unique processes known as Windows Services. Services and apps vary in that they (a) operate in the background, (b) typically don't have a user interface or communicate with the user, (c) are lengthy processes that continue until the computer shuts down, and (d) launch automatically upon restarting. Both client-side and server-side (DFS) implementations of Windows service routines were used in this project. In order to exchange data, these services also provide socket architectures for clients and servers.

4.4. Encryption

The study of using sophisticated mathematical methods to securely transfer data while maintaining the confidentiality of information is known as cryptography. Before transmission begins, the data is altered using the encryption key, and the recipient uses the proper decryption key to restore the original data. For many years, symmetric and asymmetric encryption techniques have been employed with success. Public key cryptography is another name for asymmetric encryption.

This is a very recent technique that encrypts plain text using two keys (public and private). While the private key is used for decryption, the public key is used for encryption. These two keys were made using unique mathematical methods. Everyone can know the public key, but the secret key needs to be on the server side and should not be accessible by anybody else. Despite their differences, the keys have mathematical relationships with one another. Since having the public key will not help decipher the encrypted data, the private key is kept confidential and is readily

shared with the target devices to which the data transfer is to be carried out. As asymmetric key algorithms, RSA (Rivest–Shamir–Adleman), DSA (Digital Signature Algorithm), Diffie–Hellman, and PGP (Pretty Good Privacy) are frequently employed.

The fastest, easiest to use, and most well-known method that uses a single secret key to both cipher and decrypt the associated data is symmetric cryptography, sometimes referred to as "single key cryptography." Blocks or data streams are typically encrypted in bulk using symmetric key methods. There are numerous symmetric key algorithms, including Blowfish, Stream Cipher Algorithm, Advanced Encryption Standard (AES), and Data Encryption Standard (DES). With a key length ranging from 32 to 448 bits, Blowfish is among the fastest block ciphers currently in use. The Blowfish algorithm with a key length of 128 bits was chosen for this investigation.

4.5. File Handling Routines

An operating system's primary responsibility is to maintain a reliable, quick, and effective file system. Operating systems of all kinds attempt to handle physical storage as effectively as possible by dividing it equally into 4 KB pieces known as clusters. There are essentially two distinct approaches to disk space management. The Unix-based bitmap approach is the first. Block storage devices are separated into clusters using this manner, and each cluster has a matching one bit, where 0s denote allocated blocks and 1s denote free disk blocks. The number of bits is equal to the number of blocks. The accessible, vacant disk blocks for the files that will be added to the file system are located using this map structure. Windows-based operating systems also effectively allocate disk space using this technique. This technique, known as File Allocation Table (FAT32), identifies used or empty blocks using a linked list data structure. The benefits and drawbacks of these two approaches vary. To provide a superior hybrid solution, we combine the two approaches. In the header section, we employ a bitmap structure to determine whether or not clusters are empty. A linked list-based FAT structure is utilized to keep track of the file chains; these ideas were covered in Section 4.

4.6. Programming Languages

This project made use of the Microsoft Visual Studio 2017 foundation. Additionally, the entire system was intended to be implemented using the C# programming language.

4.7. Brief System Overview and Integration

Over the past few decades, medical imaging has advanced significantly. X-rays, computed tomography (CT), molecular imaging, and magnetic resonance imaging (MRI) are among the several forms of medical imaging that have been created over time. These files are used by HIMS for data transfer, reporting, diagnosis, and treatment. The most crucial component of diagnosis and treatment are these pictures, which physicians can use to gain a better understanding of a patient's condition. A health information management system that runs on a web server with a static IP address is referred to as a client application in this study.

Three categories best describe the primary theoretical and conceptual frameworks that were employed in this study: building distributed file system architecture (the core component of our system), security concerns (for a secure connection), and client and server side service-routines. Client-server socket tools have been used to develop client-service routines (CSRs), data-node service routines (DNSRs), and client-side and server-side secure communication. Transmission blocks, seen in Figure 1, are used by these services to exchange data over TCP/IP in JSON data structures. For the client application to integrate with the system, library files (DLL files) need to be installed on the client application. No matter the size or nature of the medical picture files, the CSR is in charge of sending them to DNSR and having them read as needed. Through a secure connection, the CSR transmits the client application's requests (HIMS) to DNSR. For replica nodes to search the files, a single, smaller Windows service has also been put into place. Additionally, encrypted data transit between the other nodes is provided by this service.

The most crucial component of this work is the server side, a complete kernel service in charge of (a) listening to the designated port number for any requests from client applications; (b) verifying the CSR IP value as part of the authentication process; and (c) processing requests for reading, writing, and deleting data.

Achieving a strong security level, which comprises (a) a symmetric encryption mechanism; (b) JSON and BSON data structures for data transfer; and (c) mostly static IP values of CSR, is another crucial component of the study.
The created middle layer platform's security has significantly increased as a result of security measures.

| ISSN: 2395-7852 | www.ijarasem.com | Impact Factor: 7.583 | Bimonthly, Peer Reviewed & Referred Journal|

| Volume 12, Issue 1, January- February 2025 |

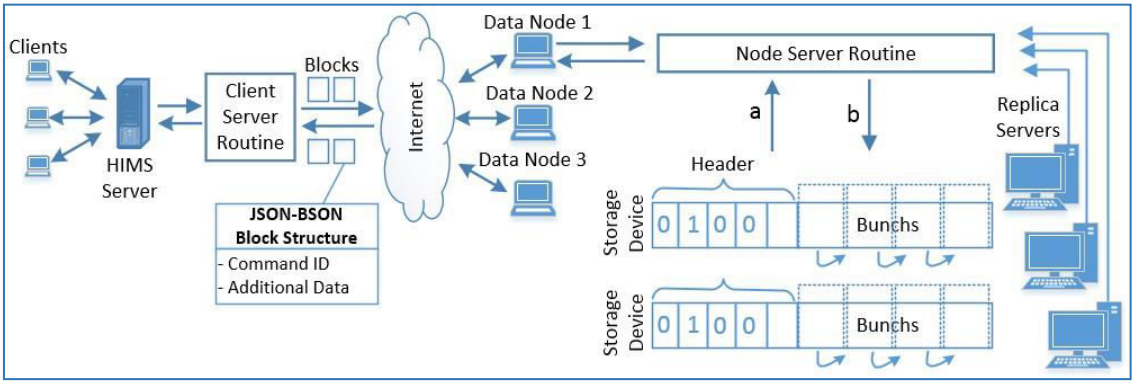


Figure 1. Windows service routines and JSON packages.

V. RESULTS AND DISCUSSION

This study established a distributed file system for medical big data sets that is fast, safe, serverless, robust, survivable, manageable, and scalable. This system, known as the Kırıkkale University Smart Distributed File System (KUSDFS), is a smart distributed file system that was created mainly for Kırıkkale University. KUSDFS, like Hadoop, NFS, and AFS, is architecturally built to handle massive volumes of data. The following provides a detailed explanation of this study's contributions. Nearly 10% of all data produced worldwide now comes from the health sector [42]. All transactions should be completed as quickly as feasible or within an acceptable delay period since healthcare software systems do not tolerate waiting from the server on which they order an operation. Conveniently completing these jobs is a strain for Hadoop and the other alternatives. as they were primarily developed to meet the needs of big data storage and to get around grid computing processes using the MapReduce framework. Therefore, the essential factor in our study is that Kırıkkale University healthcare software needs to create its own distributed file system.

The system's performance has been assessed by comparison with alternative data processing techniques. The following are the features of the comparison systems:

Hadoop Three data nodes and one name node make up the Hadoop configuration that was used for comparison. Every node is running Red Hat Enterprise Linux Server 6.0. Additionally, each node has Hadoop 2.7.2, MongoDB 3.4.2, and Java 1.6.0 installed.

CouchBase The well-known NoSQL database system Couchbase is one of the systems that are being compared. According to a study by Elizabeth Gallagher, Couchbase is unquestionably just as strong as the other well-known NoSQL databases [43]. Each bucket of the system contains 20 MB of clusters. The test computer has 200 GB of storage and 6 GB of RAM. In order to compare RDBMS database engines, Microsoft SQL Server 2014 is also chosen and installed on the same computer.
Client apps, data nodes, virtual file systems, and replica nodes are the various parts of the system.

5.1. Client Applications

Apps that get file services from the KUSDFS system are known as client apps. Since the programs use TCP/IP for all of their queries, they are platform neutral. Applications that require file services just need to have a static IP address. This system is frequently used by client programs as a "write once read many" (WORM) method for filing and archiving. Installing the Dynamic Link is necessary for any application that wishes to use KUSDFS's filing services. library on its system. The following sections provide more details on the possible methods in this system: GetAvailableDatanodeIp, ChangePassword, SaveFile, ReadFile, DeleteFile, and GetFileFromReplica. The KUSDFS data node, which is designated for the client application, is where it communicates and handles all of its operations. Symmetric cryptography techniques are used to authenticate users for secure logins that require a username and password. To improve connection security, a static client IP address is also recorded on the server. These applications can use multiple data nodes defined on KUSDFS as necessary because there is no cap on the number of client applications. This is one of the main ways that our system differs from others. The general structure of our system is shown in Figure 2.

| ISSN: 2395-7852 | www.ijarasem.com | Impact Factor: 7.583 | Bimonthly, Peer Reviewed & Referred Journal|

| Volume 12, Issue 1, January- February 2025 |

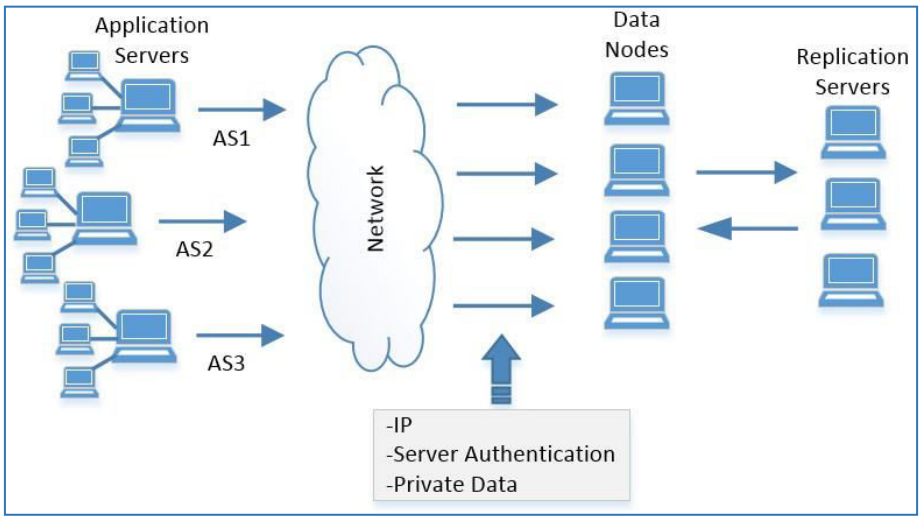


Figure 2. System overall structure.

5.2. Nodes

Node elements, of which there are two varieties—data nodes and replica nodes—are among the fundamental components of the system. Data nodes and head name nodes are typically used in distributed systems. Only the data node is used in this work because it inherits all of the head node's functionality. Data loss may result from the crushing of the head node in other systems, which typically renders the entire system inoperable [19–32]. These nodes can gather data node and server node functions, which makes them highly useful. In this sense, the system's availability and resilience are excellent. Even if there is just one node left in the entire system, it can still function. This is one of the features our system possesses.

5.3. Data Node

It is well known that data nodes are in charge of handling and keeping client-provided files. Data nodes and client applications interact through TCP/IP network services. Every data node in this project has the same priority level to process filing services for the accountable customer. Our dynamic link library software, which runs in the client, uses symmetric-key methods to establish a secure connection to data nodes. On the other hand, related data nodes serve application servers, which have static IP addresses. It is linked to the data node via this IP. Whether or not it is on the registration list, data nodes also manage the IP address of requests. Furthermore, the client has the option to safely deliver the encrypted file to the data node by encrypting the desired data using the symmetric key technique. Similarly, a high level of security has been attained with both file encryption and static IP with secure authentication. Requests from the client will be stored by the data nodes on the virtual file system, which is a sizable file on the data storage device. The system's built virtual file structure is one of its most noticeable aspects. Additionally, every data node can be used by multiple client applications. The nodes are utilized for file storage in order to ensure effective load balancing. Additionally, each data node can serve many client applications simultaneously if the client application is connected to numerous data nodes. This characteristic gives the system good availability and scalability.

5.4. System Service Architecture

Every data node has a service operating on it to carry out all of the system's functions. Using the server socket structure and the port that has been assigned to it (443, which is reserved for HTTPS, but we utilized this port), this program listens to the client applications. Upon activating the service routine:

1. It creates the device list and identifies every data storage device on the computer.
2. It generates a big, empty file that contains 80% (the ideal file size based on our test and experiment) of every device in the list (this file is only generated once for each drive). Figure defines this description.
3. A disc header and an array-based bunch list, both with a size of 10 MB, are features of the file that corresponds to a computer's disc drive. A bitmap of the bunch list is contained in the disc header, and each bunch—whether or not it is part of a file—is represented by one bit in the header. 3. This file's structure is explained as a virtual file system below.
4. When enabled, the data node reads the SystemDataFile from the system's initial node. All of the system's data nodes, replica nodes, and client apps are fully detailed in this file. It's really little for this size. Every data node receives this list from the first node with a consistent IP number upon computer startup, as indicated in Table 1. The system administrator then sends the CheckServerList function of the service routine to every data node whenever a new node is added to the system or the list is updated. Every client application is linked to the designated data node. The functions used are GetAvailableDatanodeIp, SaveFile, ReadFile, DeleteFile, CheckServerList, and GetFileFromReplica.

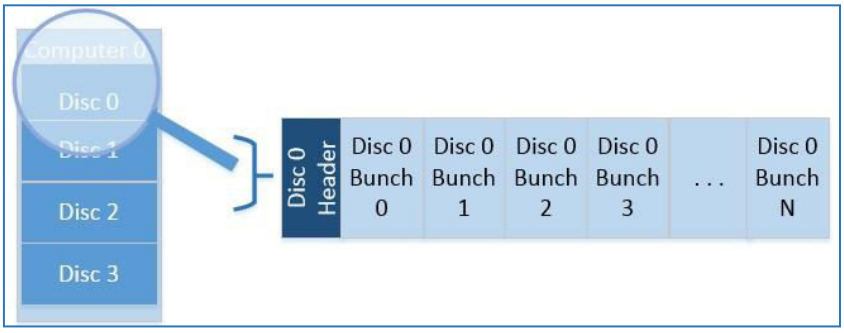


Figure 3. System data storage device architecture.

IP	Description
xxx.xxx.xxx.xxx	Data
xxx.xxx.xxx.xxx	Data
xxx.xxx.xxx.xxx	Replica
xxx.xxx.xxx.xxx	Replica
xxx.xxx.xxx.xxx	Client Application
xxx.xxx.xxx.xxx	Client Application
xxx.xxx.xxx.xxx	Client Application

5.5 SaveFile

The data node functions as a load balancer and is also in charge of the client application. The IP address of the subsequent data node in charge of file storage operations is sent by this node. As a result, there are two stages involved in saving a file. The IP value of the relevant data node is taken from the related data node by the client application when it wishes to store the file. As previously stated, a related data node indicates that client applications have a proxy node in charge of routing. Before saving the file, as seen in Figure 4, the client obtains from the proxy node the IP address of the data node that will house the file. Different data node IPs are provided for SaveFile operations by load balance feature. The GetAvailableDatanodeIP method is used to do this task. Using the SaveFile function, the system saves the client file to the data node that is almost obtained from the proxy node. The SaveFile function modifies the matching bunch in bitmap (a modified version of the Unix bitmap and Windows FAT32 structure) and saves the file in its virtual file system with a linked list structure. The client receives a distinct file name that includes the file's data node id and starting bunch number as a result of this transaction. In conclusion, the client application obtains the IP address of the data node that will store the file whenever it wishes to do so. To store the file in this manner, the client transitions to data connection with the destination data node. Data is sent to the head node in head node-structured systems, and the head node publishes the data to the data node. The amount of data sent is doubled with this model. due to the fact that information travels from the client to the head node before arriving at the data node. The number of concurrent connections to the head node is likewise decreased by this kind of activity. However, in our system, the data is sent once after the client and the data node that will store the file talk. This maximizes the number of concurrent connections.

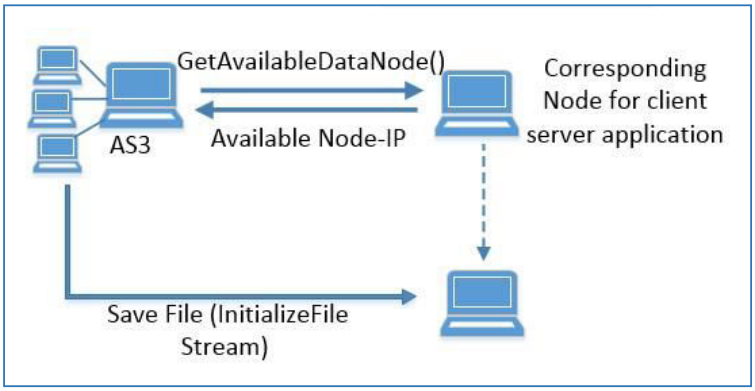


Figure 4. File storage processing.

The client that receives these values executes the requests and handles only this data node in the ReadFile and DeleteFile commands. The client application uses the GetFileFromReplica command to obtain one of the other copies of the file only in the event that the ReadFile operation fails.
The system administrator sends the CheckServerList command to every node whenever a new client is added to the system and the data nodes are altered.

5.6 Virtual File System

On the physical device, the system generates a file that takes up 80% of the available space. At this stage, the file will create a continuous region on the physical disk, as seen in Figure 3. As a result, less disk I/O will be needed. Figure 5 illustrates the bunch structure of this file, which is separated into blocks called bunches of 10 MB. A bunch is made up of 50 bytes of the file name and data block, 4 bytes of the client application IP, 1 byte of data identifying which replica server is kept, and 3 bytes of the next bunch information. The index number begins at zero, and the bunches function just like an array. The disc header contains 200 KB of bitmap data and 3 bytes of data providing the bunch size, as seen in Figure 6. The bitmap structure controls whether or not each bunch is empty. Each bunch is represented by a single bit, and a "1" indicates a used or reserved bunch; if not, the bunch is empty, indicating that it is prepared for usage in the file chain. The size of this bitmap field is 200 KB. Consequently, 16 TB of storage space is attained. A 3-byte pointer that stores the subsequent bunch within itself is stored in each bunch. The file chain is used to store files larger than a bunch size. Each bunch points to the following data block via a 3-byte next pointer, as seen in Figure 7. The index of the following bunch is stored in the pointer. The bunch indicates EOF if this pointer is set to "0," meaning it is the last bunch in the file chain. The files that are transferred to data nodes must be stored, read, deleted, and backed up. One of the simplest aspects of this structure is that the reading begins with a single disk access since the file name is delivered with the starting bunch number of the file stored in the nodes. System performance is quite good with this feature. This situation is shown in Table 2. The result of comparing the performance values of KUSDFS with other systems is shown in the diagram in Figure 8.

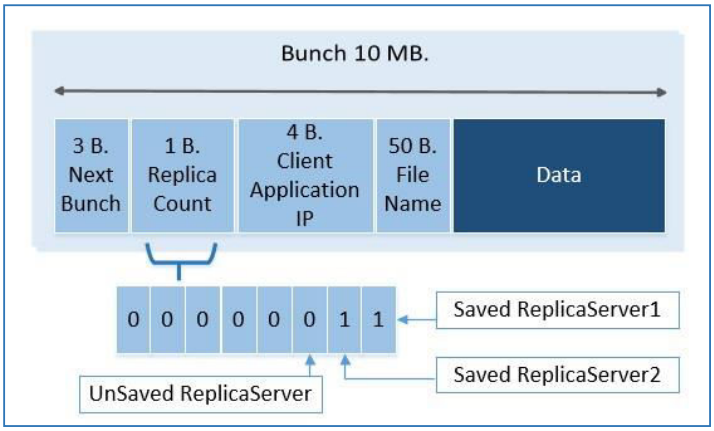


Figure 5. Bunch structure of file system.

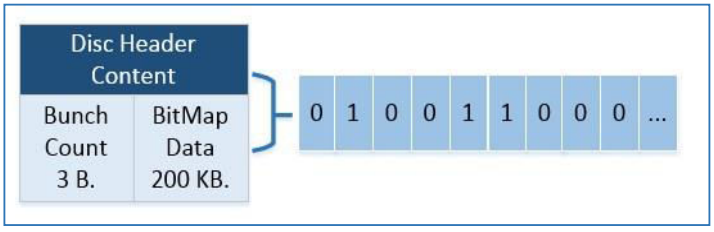
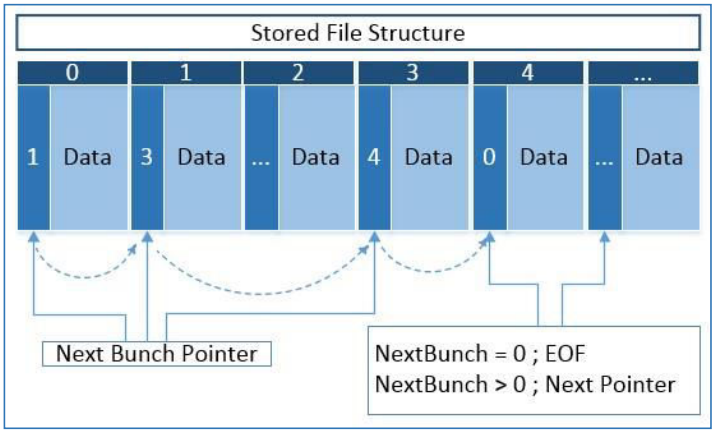


Figure 6. Structure of disc header.



File Size (KB)	KUSDFS (ms.)	OS (ms.)	NoSQL (ms.)	RDBMS (ms.)	Hadoop (ms.)
30	0.01	0.04	0.60	0.75	0.80
1000	0.92	1.43	2.74	3.15	4.01
10000	4.40	4.48	8.44	9.97	11.15
20000	4.48	11.19	18.01	18.16	20.45
30000	11.36	14.80	27.22	27.80	30.15
50000	22.60	24.54	43.95	44.08	47.88

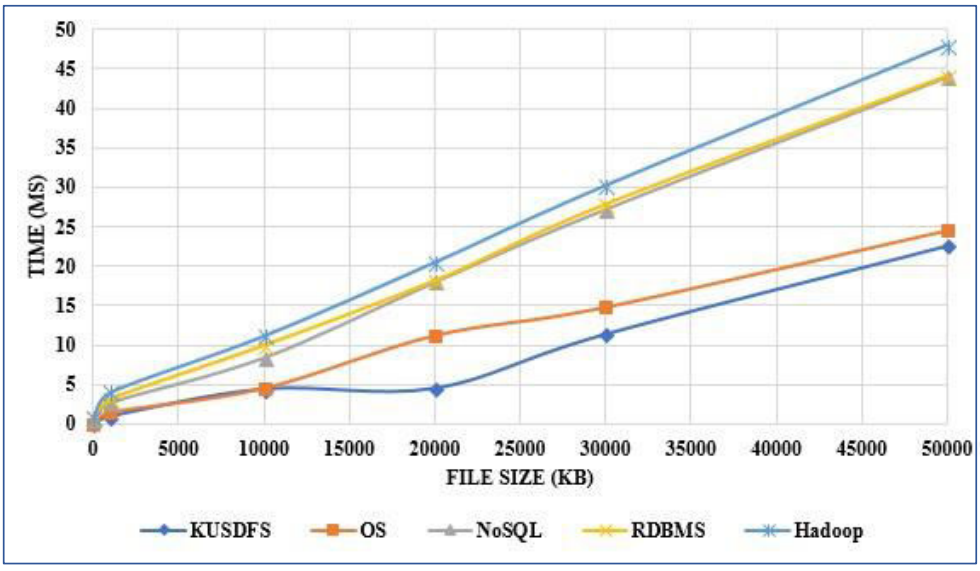


Figure 8. The response times given by the systems for different file size (ms.).

First off, several client application servers can use the built system. All queries are sent to the client application server, which is matched to one of the data nodes.
(node for data). Multiple client servers can be served by a single data node, increasing the system's resilience. Different data nodes may interface with application servers, but this will undoubtedly not have an impact on how well their systems function. Multiple client servers can be served simultaneously by each of the system's data nodes. To put it briefly, any node in the system can serve as a server upon request. Survivability, availability, and reliability scores are quite good with this feature.

When the client-server needs to store a file, it asks the data node for the address of the next data node. The client program works with the data node to which it is linked. The file is written to the data node at this IP address by the

| ISSN: 2395-7852 | www.ijarasem.com | Impact Factor: 7.583 | Bimonthly, Peer Reviewed & Referred Journal|

| Volume 12, Issue 1, January- February 2025 |

client-server that receives this address. To the system's data node, the application server writes its files in the correct order. Giving the program an IP address is the only responsibility of the data node to which it is connected. Actually, this structure is similar to the Hadoop architecture's name node.

5.7. Replica Node

Replica nodes, which are considerably simpler to process on this system, are also utilized to improve the system's fault tolerance in the DFS as an essential backup technique. Initially, these nodes are employed to create copies of the files that data nodes store. The uploaded files are sent to replica nodes asynchronously by the data nodes. These files are kept in the underlying operating system file structure by replica nodes. The operating system's filing service is employed in this case, but no unique file structure or other file processing techniques are applied. Accordingly, replica nodes function as known file servers that are only in charge of keeping files on the volume-directory structure of the operating system. For DFS, the default replica count was set at three; however, it can be increased to eight for all nodes. A service on each replica node is in charge of keeping track of, erasing, updating, forwarding, and searching the system's files. SearchFile commands are sent to all replica nodes when a file search occurs or is required in one of the data nodes or clients. At the same time, the replica nodes begin searching for the targeted file in their volume structure as grid computing, and at the conclusion of the concurrent search process, each node returns the search results indicating whether or not the search process was successful. Put differently, when a node requests to search a file, SearchFile messages are sent to all of the replicas. The replicas then operate in parallel, and the result is sent to the data node that made the request. Replica nodes process ReplicaWrite, ReplicaRead, and ReplicaDelete commands, which indicate the activities performed on a file during writing, reading, deleting, and updating on the replica, in addition to conducting file searches node, in turn.

5.8 Functional Features

1.Instead of employing name nodes, serverless architecture is chosen. The term "serverless" does not imply that there are no servers or that they are not in operation. This simply implies that we don't have to think about them as much anymore [44]. The data nodes at the same level are simultaneously regarded as serverless hierarchically. The four nodes utilized in this study support various client application kinds' file storage needs. The system will automatically direct the client to the other data nodes if any of these nodes are disabled. In this manner;

• The term "serverless" does not imply that servers are no longer used. It merely indicates that developers shouldn't give them as much thought anymore. Without needing to work within physical constraints or capacities, computing resources are employed as services.

• Several sources, such as multiple hard drives, can be used on the same computer.

2.Businesses require small and medium block sizes. Hadoop and the others are typically set up for a block size of at least 64 MB. This makes it challenging to find a great option for small and medium-sized big data problems, which leads to those not being available.

3.The issue with name node crash recovery has been resolved. Due to the absence of a name node, the system has a very high fault tolerance.

4.The underlying operating system has created a virtual file system that is simple to handle. The bunches, which have a constant size of 10 MB, were managed using a combination of bitmap and linked list structures.

5.Client programs that provide us with adequate performance gain save the file's start address.

6.Our system's most crucial feature is that every data node can function as a server for any application. To put it another way, any computer in the system has the ability to serve files.

7.Every node in the cluster has access to all IP lists that include data nodes, replica IPs, and secure connection IPs.

Non-functional specifications that KUSDFS have:

1.Performance (reading): We have attained very good performance when compared to other systems in terms of data processing speed.

2.Scalability: Because of its features, adding new nodes to the system merely requires installing the service application. The system is built to support thousands of nodes in this manner.

whereas it can function with multiple nodes according on the requirements of the institution.

3.Survivability is the ability of the system to endure and continue to perform its essential activities in spite of all the negative effects. The designed system is capable of functioning even when at least one node is active.

4.Availability: the system's ability to consistently and successfully deliver its services. The designed system has been used for this. The system has a great chance of surviving, especially given there isn't a name node.

5.Security: To increase the system's level of security, synchronous encryption and static IP addresses were employed.

6.Minimum cost: The system's data nodes are standard computers without any special features.

VI. DISCUSSION

Big data has long been a topic of intense interest in many different domains, particularly in the medical industry. Researchers have recently become more interested in leveraging the Hadoop environment to manage this issue. However, a large portion of the study to far has been descriptive in character, describing how MapReduce activities or data analytics methods are applied to the data. It has not, however, addressed what will be done for effective data storage with minimal internal fragmentation for the demands of small, medium, and large institutions.
It is intended that this study will advance our knowledge of problem-oriented stand-alone solutions.
The distinctive strengths of this study are:
1.Performance in reading and writing because of the hybrid architecture.
2.By using no-name nodes and treating each node as a server, robustness and availability are achieved.
3.ideal load distribution.
4.successful incorporation of inexpensive, common hardware components.
5.Use 128-bit symmetric cryptology to establish a secure connection.
6.Installing the library files on the node allows for simple scalability.
7.Using a block size of 10 MB makes it appropriate for healthcare facilities. The system's advantages, disadvantages, and strengths are displayed in Figure 9.

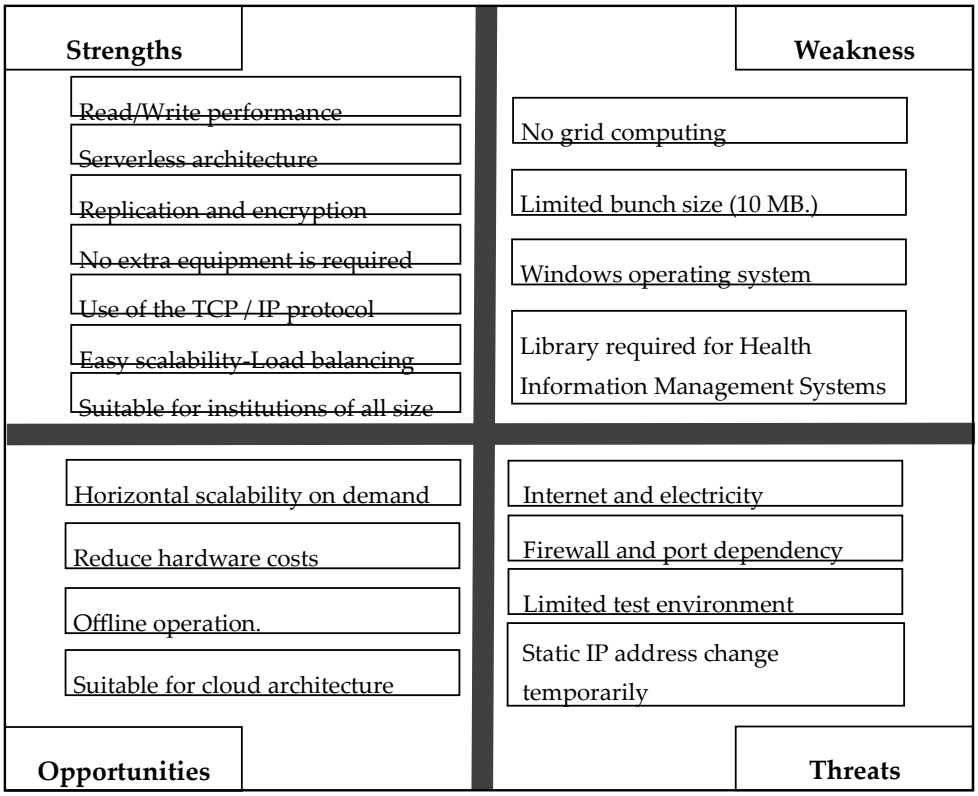


Figure 9. SWOT Analysis of the proposed system.

The limitations of the present study naturally include:
1. No grid computing infrastructure.
2. 10 MB is the fixed bunch size.
3. Static IP address changes result in brief system outages, although this only affects client applications and has no bearing on system availability.
4. Four nodes in a constrained test environment.
5. Symmetric cryptography uses a single key to function, and its security depends on the key being kept secret.

The benefits and drawbacks of this study are so examined below. Future studies will be cruciato addressing the shortcomings of this system.

| ISSN: 2395-7852 | www.ijarasem.com | Impact Factor: 7.583 | Bimonthly, Peer Reviewed & Referred Journal|

| Volume 12, Issue 1, January- February 2025 |

VII. CONCLUSIONS

In this study, we developed fast, secure, serverless, robust, survivable, easy to manage, scalable distributed file system especially for small and medium size big data sets. The designed smart distributed file system (KUSDFS) was developed for Kırıkkale University. The system is an independent system platform opposite of most distributed systems. It uses the TCP/IP protocol. Server nodes, head nodes, or name nodes are not used. The system is serverless. In this way, the survivability of the system is ensured. When a data node does not work properly in the system, other nodes can execute the requests. An unlimited number of data nodes can be added easily to the system when needed only by installing windows service routine to the node. It is the superiority of the system compared with other systems.

Similar to other distributed file systems, system security is at an acceptable level. There are two methods to get this. Encrypting the data that the application software delivers to the data nodes is the second step, after verifying the IP addresses of the client computers that are serviced by the data node.
Our system has better load balance performance than existing distributed file systems since a data node can serve several clients. Similarly, a client has the ability to upload data to several data nodes.

The "bunch" and "disc header" data sets make up a disk in the system that was built. The number of bunches on the disc and whether the bunch is empty using the bitmap structure are stored in the disc header. A bunch contains the data itself, the IP of the client application that loads the data, the next bunch, and the data replicas. Each bunch can refer to the subsequent bunch that is a part of the file chain using the file system that has been designed.

The operating system file operations system calls—known as APIs in Windows—are used in the replication process. Asynchronously, data nodes communicate the files they have uploaded to themselves to the replica nodes.

The designed distributed file system is compared with other filing systems. According to of our analysis, our system performed 97% better than the NoSQL system, 80% better than the RDBMS, and 74% better than the operating system.

KUSDFS may eventually integrate with Kırıkkale province's healthcare facilities. The institutions' expenses, workload, and technological requirements will be at their lowest at this stage. By simply installing library files, users can access and integrate with the system without creating their own archiving systems. Healthcare-specific, the designed system has a set bunch size of 10 MB. Future research may build a dynamic bunch size to accommodate disparate applications with varying file sizes and types. Functionalities from grid computing might also be added to the system to increase its processing speed and power. Future research on this topic is crucial. Furthermore, GPUs, which have been increasingly popular in recent years, can be incorporated into the system to improve grid performance and provide a more specialized distributed computing platform. To prove this property, however, more research on this subject is needed.

Contributions from the Author: A.E. designed and organized the system. M.Ü. conducted a literature search, assisted with experiments, and examined test results. The mechanism was put into place by both authors, who also wrote the paper, reviewed, and approved the text that was submitted.

ACKNOWLEDGMENT

The researcher expresses their sincere gratitude to all individuals and institutions that contributed to the success of the study, *Creating a File System Architecture to Address Health Large Data Archiving and Storage Issues with a Distributed File Systems.* They thank the experts and participants for sharing valuable insights into challenges and advancements in network administration, which were essential in shaping the study.

Appreciation is also extended to Surigao del Norte State University for its facilities ang resources, as well as to the faculty, staff, and advisors for their support, encouragement, and constructive feedback.

Lastly, they acknowledge the authors and researchers whose works laid this comparative analysis's foundation. This study would not have been possible without these contributions.

| ISSN: 2395-7852 | www.ijarasem.com | Impact Factor: 7.583 | Bimonthly, Peer Reviewed & Referred Journal|

| Volume 12, Issue 1, January- February 2025 |

REFERENCES

1. Internet Live State. Available online: http://www.internetlivestats.com (accessed on 16 July 2016).
2. Special Reports. Available online: http://wearesocial.com/uk/special-reports/digital-in-2016 (accessed on 27 June 2016).
3. Internet World Stats. Available online: https://www.internetworldstats.com/emarketing.htm (accessed on 21 May 2018).
4. Facebook Data Warehouse. Available online: https://code.facebook.com/posts/229861827208629/scalingthe-facebook-data-warehouse-to-300-pb/ (accessed on 27 June 2016).
5. Dhavalchandra, P.; Jignasu, M.; Amit, R. Big Data—A Survey of Big Data Technologies. *Int. J. Sci. Res. Technol* **2016**, *2*, 45–50.
6. Dean, B.B. Use of Electronic Medical Records for Health Outcomes Research. *Med. Care Res. Rev.* **2009**, *66*, 611–638. [CrossRef] [PubMed]
7. Erguzen, A.; Erdal, E. Medical Image Archiving System Implementation with Lossless Region of Interest and Optical Character Recognition. *J. Med. Imag. Health Inform.* **2017**, *7*, 1246–1252. [CrossRef]
8. Dinov, I.D. Volume and Value of Big Healthcare Data. *J. Med. Stat. Inform.* **2016**, *4*, 3. [CrossRef] [PubMed]
9. Elgendy, N.; Elragal, A. Big Data Analytics: A Literature Review Paper. In Proceedings of the 14th Industrial Conference (ICDM 2014), St. Petersburg, Russia, 16–20 July 2014.
10. Gürsakal, N. *Büyük Veri*; Dora Publishing, 2. Press: Bursa, Turkey, 2014.
11. Turkish Republic, Ministry of Health. *Sag˘lık Istatistikleri Yıllıg˘ı 2016˙* ; Turkish Republic, Ministry of Health: Ankara, Turkey, 2016.
12. Raghupathi, W.; Raghupathi, V. Big data analytics in healthcare: Promise and potential. *Health Inform. Sci. Syst.* **2014**, *2*, 3. [CrossRef] [PubMed]
13. Klein, J.; Gorton, I.; Ernst, N.; Donohoe, P.; Pham, K.; Matser, C. Application-Specific Evaluation of NoSQL Databases. In Proceedings of the 2015 IEEE International Congress on Big Data (BigData Congress), New York City, NY, USA, 29 October–1 November 2015.
14. Kodcu. Available online: https://blog.kodcu.com/2014/03/nosql-nedir-avantajlari-ve-dezavantajlarihakkinda-bilgi/ (accessed on 13 June 2017).
15. He, H.; Du, Z.; Zhang, W.; Chen, A. Optimization strategy of Hadoop small file storage for big data in healthcare. *J. Supercomput.* **2016**, *72*, 3696–3707. [CrossRef]
16. Wanga, Y.; Ann, L.; Terry, K.; Byrd, A. Big data analytics: Understanding its capabilities and potential benefits for healthcare organizations. *Technol. Forecast. Soc. Chang.* **2018**, *126*, 3–13. [CrossRef]
17. Mishra, S. A Review on Big Data Analytics in Medical Imaging. *Int. J. Comput. Eng. Appl.* **2018**, *12*, 31–37.
18. Sobhy, D.; El-Sonbaty, Y.; Elnasr, M.A. MedCloud: Healthcare Cloud Computing System. In Proceedings of the 2012 International Conference for Internet Technology and Secured Transactions, London, UK, 10–12 December 2012.
19. Alsberg, P.A.; Day, J.D. A Principle for Resilient Sharing of Distributed Resources. In Proceedings of the 2nd International Conference on Software Engineering (ICSE ’76), San Francisco, CA, USA, 13–15 October 1976.
20. Ellis, C.A.; Floyd, R.A. The Roe File System. In Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems, Clearwater Beach, FL, USA, 17–19 October 1983.
21. Popek, G.; Walker, B.; Chow, J.; Edwards, D. LOCUS a network transparent, high reliability distributed system. In Proceedings of the Eighth ACM Symposium on Operating Systems Principles (SOSP ’81), Pacific Grove, CA, USA, 14–16 December 1981.
22. Sandberg, R.; Goldberg, D.; Kleiman, S.; Walsh, D.; Lyon, B. Design and Implemantation of the Sun Network File System. In Proceedings of the USENIX Conference and Exhibition, Portland, OR, USA, 11–14 June 1985.
23. Coulouris, G.; Dollimore, J.; Kindberg, T.; Blair, G. *Distributed Systems: Concepts and Design*, 5th ed.; Pearson Education Limited: Essex, UK, 2011.
24. Heidl, S. Evaluierung von AFS/OpenAFS als Clusterdateisystem, Berlin: Technical Report, Zuse-Institut Berlin. 2001. Available online: http://www2.cs.upb.de/StaffWeb/jens/Courses/VLZIB/Datamngmnt/ausarbeitung_sebastian_afs.pdf (accessed on 18 December 2008).
25. Bzoch, P.; Šafarik, J. Algorithms for Increasing Performance in Distributed File Systems. *Acta Electrotech. Inform.* **2012**, *12*, 24–30. [CrossRef]
26. Karasulu, B.; Korukog˘lu, S. Modern Dag˘ıtık Dosya Sistemlerinin Yapısal Kars¸ılas¸tırılması. In Proceedings of the Akademik Bilis¸im’2008, Çanakkale, Turkey, 30 January–1 February 2008.
27. Thekkath, C.A.; Mann, T.; Lee, E.K. Frangipani: A scalable distributed file system. In Proceedings of the Sixteenth ACM symposium on Operating systems principles (SOSP ’97), Saint Malo, France, 5–8 October 1997.
28. Adya, A.; Bolosky, W.J.; Castro, M.; Cermak, G.; Chaiken, R.; Douceur, J.R.; Howell, J.; Lorch, J.R.; Theimer, M.; Wattenhofe, R.P. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In Proceedings of the 5th Symposium on Operating Systems (USENIX), Boston, MA, USA, 9–11 December 2002.

29. Weil, S.A.; Brandt, S.A.; Miller, E.L.; Long, D.D.E.; Maltzahn, C. Ceph: A Scalable, High-Performance Distributed File System. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), Seattle, WA, USA, 6–8 November 2006.
30. Shvachko, K.; Kuang, H.; Radia, S. The Hadoop Distributed File System. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, 3–7 May 2010.
31. Yavuz, G.; Aytekin, S.; Akçay, M. *Apache Hadoop Ve Dag˘ıtık Sistemler Üzerindeki Rolü*; Dumlupınar Üniversitesi, Fen Bilimleri Enstitüsü Dergisi: Kütahya, Turkey, 2012; pp. 43–54.
32. Wikipedia. Available online: https://en.wikipedia.org/wiki/Apache_Hadoop (accessed on 27 June 2016).
33. Thomson, A.; Abadi, D.J. CalvinFS: Consistent WAN Replication and Scalable Metadata Management for Distributed File Systems. In Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15), Santa Clara, CA, USA, 16–19 February 2015.
34. Al-Kahtani, M.S.; Karim, L. An Efficient Distributed Algorithm for Big Data Processing. *Arab. J. Sci. Eng.* **2017**, *42*, 3149–3157. [CrossRef]
35. Cuares, F., & Teleron, J. I. (2024). Harmony in nodes: Exploring efficiency and resilience in distributed systems. *Engineering and Technology Journal, 9*(5). https://doi.org/10.47191/etj/v9i05.32
36. Wang, Q.; Wang, H.; Zhang, C.; Wang, W.; Chen, Z.; Xu, F. A parallel implementation of idea graph to extract rare chances from big data. In Proceedings of the IEEE International Conference on Data Mining Workshop, Shenzhen, China, 14 December 2014.
37. Liang, Z.; Li, W.; Li, Y. A parallel probabilistic latent semantic analysis method on MapReduce platform. In Proceedings of the 2013 IEEE International Conference on Information and Automation (ICIA), Yinchuan, China, 26–28 August 2013.
38. Chen, T.; Wei, H.; Wei, M.; Chen, Y.; Hsu, T.; Shih, W. LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. In Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS), San Diego, CA, USA, 20–24 May 2013.
39. Muja, M.; Lowe, D. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *36*, 2227–2240. [CrossRef] [PubMed]
40. Maderazo, R. T., & Teleron, J. I. (2024). Decentralized architectures: A paradigm for scalable, fault-tolerant, and efficient distributed systems. *Engineering and Technology Journal, 9*(2). https://doi.org/10.47191/etj/v9i02.16
41. Lu, F.; Hong, L.; Changfeng, L. The improvement and implementation of distributed item-based collaborative filtering algorithm on Hadoop. In Proceedings of the 2015 34th Chinese Control Conference (CCC), Hangzhou, China, 28–30 July 2015.
42. Cevher, V.; Becker, S.; Schmidt, M. Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics. *IEEE Signal Process. Mag.* **2014**, *31*, 32–43. [CrossRef]
43. Yang, J.; Tang, D.; Zhou, Y. A Distributed Storage Model for EHR Based on Hbase. In Proceedings of the 2011 International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII), Shenzhen, China, 26–27 November 2011.
44. Hossaina, M.S.; Muhammad, G. Cloud-assisted Industrial Internet of Things (IIoT)—Enabled. *Comput. Netw.* **2016**, *101*, 192–202. [CrossRef]
45. Chandra, D.G. BASE analysis of NoSQL database. *Future Gener. Comput. Syst.* **2015**, *52*, 13–21. [CrossRef]
46. Jalaman, J. R. C., & Teleron, J. I. (2025). *Optimizing operating system performance through advanced memory management techniques: A comprehensive study and implementation.* Retrieved January 28, 2025, Researchgate
47. Redwrite. Available online: http://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-isserverless/ (accessed on 15 June 2017).
48. Gosela, R. R. U., & Teleron, J. I. (2023). Revolutionizing connectivity: Advancing equitable traffic access through advanced scalability and load balancing network architecture via adaptive equalization algorithms. *International Journal of Advanced Research in Science Communication and Technology.* https://doi.org/10.48175/IJARSCT-14026

International Journal of Advanced Research in Arts, Science, Engineering & Management (IJARASEM)

| Mobile No: +91-9940572462 | Whatsapp: +91-9940572462 | ijarasem@gmail.com |