applied sciences

MDPI

Article

Developing a File System Structure to Solve Healthy Big Data Storage and Archiving Problems Using a Distributed File System

Atilla Ergüzen and Mahmut Ünver *

Department of Computer Engineering, Faculty of Engineering, Kırıkkale University Ankara Yolu 7. Km, 71450 Kırıkkale, Turkey; atilla@kku.edu.tr

* Correspondence: munver@kku.edu.tr; Tel.: +90-543-780-3069

Received: 7 May 2018; Accepted: 30 May 2018; Published: 2 June 2018

Featured Application: The most important key features of this study are high performance, easy scalability and serverless architecture. In this way, the system can work with fewer hardware elements and be more robust than others that use name node architecture. Also, both the reliability and performance of the system are significantly increased by separating replication nodes from data nodes. As a result, a complete big data solution that is easy to manage and performs well has been produced and successfully used.

Abstract: Recently, the use of internet has become widespread, increasing the use of mobile phones, tablets, computers, Internet of Things (IoT) devices and other digital sources. In the health sector with the help of new generation digital medical equipment, this digital world also has tended to grow in an unpredictable way in that it has nearly 10% of the global wide data itself and continues to keep grow beyond what the other sectors have. This progress has greatly enlarged the amount of produced data which cannot be resolved with conventional methods. In this work, an efficient model for the storage of medical images using a distributed file system structure has been developed. With this work, a robust, available, scalable, and serverless solution structure has been produced, especially for storing large amounts of data in the medical field. Furthermore, the security level of the system is extreme by use of static Internet protocol (IP), user credentials, and synchronously encrypted file contents. One of the most important key features of the system is high performance and easy scalability. In this way, the system can work with fewer hardware elements and be more robust than others that use name node architecture. According to the test results, it is seen that the performance of the designed system is better than 97% from a Not Only Structured Query Language (NoSQL) system, 80% from a relational database management system (RDBMS), and 74% from an operating system (OS).

Keywords: big data; distributed file system; health data; medical imaging

1. Introduction

In recent years, advances in information technology have increased worldwide; internet usage has exponentially accelerated the amount of data generated in all fields. The number of internet users was 16 million in 1995. This number reached 304 million in 2000, 888 million in 2005, 1.996 billion in 2010, 3.270 billion in 2015, and 3.885 billion in 2017 [1–3]. Every day, 2.5 exabytes (EB) of data are produced worldwide. Also, 90% of globally generated data has been produced since 2015. The data generated are in many different fields such as aviation, meteorology, IoT applications, health, and energy sectors. Likewise, the data produced through social media has reached enormous volumes. Not only does

Facebook.com store 600 terabytes (TB) of data a day in 2014 but Google also processed hundreds of petabytes (PB) of data per day in the same year [4,5]. Data production has also increased at a remarkable rate in the health sector widespread use of digital medical imaging peripherals has triggered this data production. Also, the data generated in the health sector has been reached a such a point that it cannot be managed easily with traditional data management tools and hardware. Healthcare has accumulated a big data volume by keeping patients' records, creating medical imaging that helps doctors with diagnosis, and through the output of various devices in digital format and storing the results of different surveys. Different types of data sources produce data in various structured and unstructured formats—such as patient information, laboratory results, X-ray devices, computed tomography (CT)s and magnetic resonance imaging (MRI). World population and average human lifespan is apparently increasing continuously, which means an exponential increase in the number of patients to be served. As the number of patients increases, the amount of collected data also increases dramatically. Besides, exhaustive digital healthcare devices make higher-density graphical outputs easy adds to the growing body of data. In 2011, the data amount of the healthcare sector in the U.S. reached 150 EB. In 2013, it appeared to have achieved 153 EB. In 2020, it is estimated that this number will reach 2.3 ZB. For example, in Electronic Medical Record (EMR) has increased 31% from 2001 to 2005 and more than 50% from 2005 to 2008 [6,7]. While neuroimaging operation data sizes have reached approximately to 200 GB per year between 1985 and 1989, indeed it has risen to 5 PB annually between 2010 and 2014 which is an indicator of the data increase in the health sector [8].

In this way, new problems have emerged due to the increasing volume of data generated in all fields at the global level. Now there are substantial challenges to store and to analyze the data. The storage of data has become costlier than gathering it [9]. Thus, the amount of data that is produced, stored, and manipulated has increased dramatically, and because of this increase, 'big data' and data science/knowledge have begun to develop [10]. 'big data' consists of three concepts: variety, velocity, and volume; concerning healthcare records, finding an acceptable approach to cover these issues is particularly difficult to accomplish.

Big data problems in healthcare and the objectives of the study according to the previous arguments are listed as follows:

1. Increasing number of patients: the global population and average human lifespan are apparently increasing. For example, in Turkey, the number of visits to a physician has increased by about 4% per year since 2012 [11]. Moreover, the total number of per capita visits to a physician in Health Care Facilities in 2016 was 8.6 while this value was 8.2 in 2012. As the number of patients increases, the amount of collected data also increases dramatically which raises much more data to be managed.
2. Technological devices: exhaustive digital healthcare devices create high-resolution graphical outputs which means the huge amount of data to be stored.
3. Expert personnel needs: to manage big data in institutions by using Hadoop, Spark, Kubernetes, Elasticsearch, etc., qualified information technology specialists must be occupied to deploy, manage, and store big data solutions [12].
4. Small file size problem: current solutions for healthcare including Hadoop based solutions have a block size of 64 MB which will be detailed in the next section; this leads to vulnerabilities in performance and unnecessary storage usage, called 'internal fragmentation', that is difficult to resolve.
5. Hospital Information Management Systems (HIMS): These systems are comprehensive software and related tools that help healthcare providers produce, store, fetch, and exchange patient information more efficiently and enable better patient tracking and care. HIMS must have essential non-functional properties like (a) robustness, (b) performance, (c) scalability, and (d) availability. These properties basically depend on constructed data management architecture which includes configured hardware devices and installed software tools. HIMS is responsible for solving big data problems alone. HIMS is much more than an IT project or a traditional

application software. So, third-party software tools are needed to achieve the objectives of the healthcare providers.

This study seeks to obtain middle layer software platform which will help to address these healthcare gaps. In other words, we have implemented a server-cluster platform to store and to return health digital image data. It acts as a bridge between HIMS and various hardware resources located on the network. There are five primary aims of this study:

1. To overcome growing data problem by implementing distributed data layer between HIMS and server-cluster platform.
2. To reduce operational costs, in this solution there is no need to employ IT specialists to install and to deploy popular big data solutions.
3. To implement a new distributed file system architecture to achieve non-functional properties like performance, security, and scalability issues which are of crucial importance for HIMS.
4. To show and to prove that there can be different successful big data solutions.
5. Especially, to solve these gaps efficiently for our university HIMS.

In this study, the first section describes general data processing methods. The second part is the literature work on the subject; the third part is the materials and methods section that describes the implemented approach. The last section is the conclusion of the evaluation that emerges as the result of our work.

2. Big Data Architecture in Medicine

Big data solutions, in healthcare worldwide, primarily consist of three different solutions:

The first is a database system which has two different popular application architectures; relational database management systems (RDBMS) and NoSQL database systems. RDBMSs, as the most widely known and used systems for this purpose, are storage methods in which data is stored in a structured format. The data to be processed must be the appropriate type and format. In these systems, a single database can serve multiple users and applications. Since these systems are built on vertical growth functionality, the data structure must be defined in advance, and they have a lot of constraints like atomicity, consistency, isolation, and durability. The strict rules that make these systems indispensable are beginning to be questioned today. However, due to the used hardware and software, the initial installation costs are high. Especially when the volume of data increases, the horizontal scalability feature will be quite unsatisfactory and difficult to manage, which is a major factor of their not being a part of big data problem resolution. Also, these systems are more complex than file systems, which most importantly is not suitable for big data. Due to the deficiency of managing RDBMSs' big data, NoSQL database systems have emerged as an alternative. The main purpose of these systems is to be able to store the increasing data volumes of the internet and to respond to the needs of high-traffic systems via unstructured or semi-structured formats. NoSQL databases are systems that provide high accessibility according to RDBMSs and in which data are easily scaled horizontally [13]. Reading and writing performances may be more acceptable than RDBMS. One of the most important features is that they are horizontally expandable. Thousands of servers can work together as a cluster and operate on big data. They are easy to program and manage due to their flexible structures. Another feature of these systems is that they must be doing grid computing in clusters that consist of many machines connected to a network, in this way, data process speeds have increased. Besides, it does not yet have as advanced features as RDBMS on data security. Some NoSQL projects are lacking in documentation and professional technical support. Also, the concept of transactions is not available in NoSQL database systems, loss of data may occur, so they are not suitable for use in banking and financial systems [14].

Basic file management functions of operating systems (OS) are used for the second solution which are called 'file servers' in literature. In this system, medical image data is stored in files and

folders in underlying operating system file structure. Operating systems use a hierarchical file system. In this structure, the files are in a tree structure, called a 'directory'. File servers store the files in the way that is determined by HIMS according to image type, file creation date, policlinic name, and patient information. HIMS executes read and write operations by invoking system-calls which provide low-level interface to storage devices with the help of the operating system. The advantages of using file servers are that they are simple to deliver, easy to implement, and have acceptable file operation performance. Writing, deleting, reading, and searching files on the operating system is a fairly fast process because the operating system has been specialized for file management. Especially, operating systems has more flexibility and performance RDBMS and NoSQL systems. However, the OS could not use these advantages to be a satisfactory solution model for big data because of the lack of horizontal scalability. The main task of OS file management is to serve system calls to other applications. So, the OS is a part of the solution rather the solution alone. Besides the storage of data not being as secure as other methods, data cannot be scaled according to data size, and backup and file transfer cannot be done safely. It seems that the operating system alone is not suitable for solving big data problems.

The third method is Distributed File System (DFS), or 'distributed file systems'. These systems are the most up-to-date systems that suppose the machines in various locations as a single framework and provide the most appropriate solution to the big data problem. Hadoop DFS and MapReduce are primarily used for big data storage and analytics. Hybrid solutions including Hadoop and NoSQL also is used and criticized in the literature. However, there are some drawbacks to using Hadoop ecosystem in the healthcare. The first one is the small files problem which means Hadoop cannot store and manage this type of files efficiently [15]. Because Hadoop is primarily designed to manage large files greater than or equal to 64 MB, this size also default block size of Hadoop clusters [15]. For example, a file that is 1 GB in size consisting of 16 blocks of 64 MB Hadoop blocks occupies 2.4 KB space in a name node. However, 100,000 files of 100 KB occupies 1 GB space in data nodes, 1.5 MB in name node. This means more MapReduce operations required when processing small files. The healthcare sector average medical image file size is 10 MB, and when this situation is taken into consideration, a new DFS system is needed to embrace systems that have large numbers of small files. This study proposes a new methodology for this issue by small block size and no name node structure. WAN at al. has identified five different strategies for how big data analytics can be effectively used to create business value for healthcare providers. This work was carried out for a total of 26 health institutions that uses Hadoop/MapReduce from Europe and the U.S. [16]. However, it is also stated in this study that with a growing amount of unstructured data, more comprehensive analytic techniques are required to be satisfied like deep learning algorithms. A significant analysis and discussion on Hadoop/MapReduce, Storm frameworks for big data in health care was presented by Liu and Park. It stated that Hadoop/MapReduce cannot be used in real-time systems due to a performance gap. Therefore, they have proposed a novel health service model called BDeHS architecture which has three key benefits. Whereas, Spark and Storm can be used more effectively for real-time data analytics of large databases according to MapReduce [17]. One study provided detailed information about the architectural design of a personal health record system called "MedCloud" constructed on Hadoop's ecosystem [18]. Another study by Erguzen and Erdal looked at big data in healthcare, a new file structure and achieving system has been developed to store regions of interest (ROIs) from MRIs. In other words, they extracted the ROI portions, which contained vital information about the patient, from the image, discarded the rest—called non-ROIs—and stored the ROIs in the newly designed file structure with a success ratio of approximately 30%. However, this work was done only to decrease the image sizes, not to effectively store big data on DFS [7]. The other study conducted by Raghupathi and Raghupathi showed that the Hadoop ecosystem has significant drawbacks for medium or big size healthcare providers: (a) it requires a great deal of programming skills for data analytics tasks using MapReduce; and (b) it is almost difficult to install, to configure, to manage Hadoop ecosystem completely, so it does not seem to be a feasible solution for medium- or large-scale healthcare providers [12].

Today, Hadoop is one of the enterprise-scaled open source solutions that makes it possible to store big data with thousands of data nodes, as well as analyze the data with MapReduce. However, there are three disadvantages to Hadoop. The first one is: Hadoop's default block size is 64 MB which presents an obstacle in managing numerous small files [15]. When a file smaller than 64 MB will be embedded in a 64 MB Hadoop block, it will cause a gap which is called internal fragmentation. On our system, the block size is 10 MB which was constructed according to the average MRI file size and it means less internal fragmentation. Secondly, the performance issue is a big problem when the system needs to run in a real-time environment. The third issue with Hadoop is that it requires professional support to construct, operate, and maintain the system properly. These drawbacks are the key factors of why we developed this potential solution. So, an original distributed file system has been developed for storing and managing healthy big data. The developed system has been shown to be quite successful for applications that run in the form of write once read many (WORM), which is a model that has many uses, such as in electronic record management system and the healthcare sector.

3. Related Studies

Big data is data that cannot be stored and administrated on only one computer. Today, to administrate it, computers connected to a distributed file system and working together in a network are used. DFSs are separated into clusters consisting of nodes. Performance, data security, scalability, availability, easy accessibility, robustness, and reliability are the most important features of big data. Big data management problems are solved by using DFS and network infrastructure. DFS-related works began in the 1970s [19]. One of the first studies is The Roe File System and it was developed for replica consistency, easy setup, secure file authorization, and network transparency [20].

LOCUS, developed in 1981, is another DFS that features network transparency, high performance, and high reliability [21]. Network file system (NFS) started to be developed by Sun Microsystems in 1984. This system is the most operated DFS on UNIX. Remote procedure call (RPC) is used for communication [22]. It is designed for enabling Unix file system to function as a 'distributed' system. The virtual file system is used as a layer. Therefore, clients can run different file systems easily and fault tolerance is high in the NFS. File status information is kept, and when an error occurs, the client reports this error status to the server immediately. File replication is not done in NFS, whereas the whole system is replicated [23]. Only the file system is shared on NFS, no printer or modem can be shared. Objects to be shared can be a unit of a directory as well as a file. It is not necessary to set up every application on a local disk in NFS, and there can be shared by using the server. For all that the same computer can be both a server and a client. As a result, NFS reduces data storage costs.

The Andrew file system (AFS-1983) and its successor's CODA (1992) and OpenAFS [24] are open sources for distributed file systems. These systems have scalable and larger cluster size. Also, they can reduce server load and cache the whole file. CODA replicates on multiple servers to increase accessibility. Whereas AFS only supports Unix, OpenAFS and CODA support MacOS and Microsoft Windows. In these systems, the same namespace is created for all clients. However, replication is limited, and read-one/write-all (ROWA) schema is used for it [25,26].

Frangipani was developed as a new distributed file system which is designed as two layers in 1997. The bottom layer consists of virtual disks. These provide storage services. It can be scaled and managed which is done automatically. On the top layer, there are several machines that use the Frangipani file system. These machines run distributed on the shared virtual disk. The Frangipani file system provides consistent and shared access to the same set of files. As the data used in the system grows, more storage space and higher performance hardware elements are needed. If one of the system components does not work, it continues to serve due to its availability. As the system grows, the added components do not make management complicated and thus there is less need for human management [27].

FARSITE (2002) is a serverless file system that runs distributed on a network. It runs distributed on a network of physically unreliable computers. The system is a serverless, distributed file system.

It does not require centralized management. Thus, there are not staff costs like a server system. FARSITE is designed to support the file I/O workload of a desktop computer in the university or a large company. It provides reasonable performance using client caching, availability, and accessibility using replication, authentication using encryption, and scalability using namespace delegation. One of the most important design goals of FARSITE is to use the benefits of the Byzantine fault-tolerance [28].

Described in 2006, the CEPH file system is located on a top layer of similar systems that do object storage. This layer separates data and metadata management. This is accomplished by the random data distribution function (CRUSH), which is designed for unreliable object storage devices (OSDs). This function replaces the file allocation table. With CEPH, distributed data replication, error detection, and recovery operations are transferred to object storage devices are running on the local file system. Thus, system performance is enhanced. A distributed set of metadata makes its management extremely efficient. The Reliable autonomic distributed object store (RADOS) layer manages all filing processes. Measurements were performed under various workloads to test the performance of CEPH, which can also work within different discs size. As a result, I/O performance is extremely high. It has been shown to have a scalable metadata management. Because of the measurements, it supports 250,000 meta transactions per seconds. With CEPH, a high performance, reliable, and scalable distributed file system has been developed [29].

In 2007, Hadoop was developed, consisting of Hadoop distributed file system (HDFS) and MapReduce-parallel computing tool. Hadoop is a framework that provides analysis and transformation of very large datasets. HDFS distributes big data by dividing into clusters on standard servers. To ensure data security backs the blocks up on the servers by copying them [30]. Hadoop/MapReduce is used to process and manage big data. The 'map' function distributes the data on the cluster and provides it to be processed. The 'reduce' function ensures the data will be combined. Hadoop has scalability, and it can be easily handle PBs of data [31]. Today, Hadoop is used by many major companies. It is preferred in industrial and academic fields. Companies like LinkedIn, eBay, AOL, Alibaba, Yahoo, Facebook, Adobe, and IBM, use Hadoop generally [32].

Announced in 2015, CalvinFS is a file system that can be scalable and has replica property using a highly efficient database designed for metadata management. For this, it divides the metadata horizontally across multiple nodes. The file operations that need to edit the metadata item works distributed. This system also supports standard file systems. This file system approach has shown that scaling can be done up to billions of files. While reducing reading delays, it can conduct hundreds of thousands of updates and millions of reads per second at the same time [33].

In 2016, Mohammed S. Al-Kahtani and Lutbul Karim presented a scalable distributed system framework [34]. The system performs scaling on the central server. The proposed framework transfers the data processing work to other computers, by the server as the amount of data collected increases. In other words, the system works distributed when several of data increases. Other frameworks like this work are: IdeaGraph algorithm [35], probabilistic latent semantic analysis (PLSA) [36], locality-aware scheduling algorithm [37], nearest neighbor algorithm [38], item-based collaborative filtering algorithm [39], recommendation algorithm [40], convex optimization [41], and parallel two-pass MDL (PTP-MDL) [40].

Yang Jin et al. designed an efficient storage table for electronic health records. The system also makes analysis and produce detailed statistical values by using MapReduce. It is designed with HBase, a distributed-column based database that runs on the Hadoop distributed file system MapReduce framework. The model is low cost and has two name nodes. In addition, HMaster and HRegionServer allow load balancing to get better performance. However, it has been noted that the system should work to develop data block replication strategies for HDFS [41].

Today, distributed file systems can be considered/examined in two main categories:

- Big data storage: Use necessary file system and cluster schema to save big data (data save).
- Big data analytics: The short and consistent analysis of the data collected from the nodes by grid computing tools (data mining).

4. Materials and Method

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

4.1. TCP/IP Protocol

Protocols specify strict rules for exchanging messages between the communicating entities across a local area network or wide area network. Each computer or mobile device on the Internet has one unique IP address that cannot overlap other devices on the Internet. The Internet protocol (IP) uses this IP addresses to connect endpoints that is a combination of an IP address and a port number by using packets or datagrams which contains source and destination device IP addresses and related information. This IP needs transport layer protocols such as 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). TCP and UDP are the most-used in internet connections. TCP has many advantages over UDP, including (a) reliability, (b) ordered transmission of packets, and (c) streaming. Therefore, TCP/IP is used in this project.

4.2. Sockets

A TCP socket is an endpoint of process defined by an IP address and a port number for client/server interaction. Therefore, TCP socket is not a connection, it is just one endpoint of a two-way communication link between two processes. There are two types of socket—client and server—which both can send and receive. So, the distinction between them is in how the connection is created. Client sockets initialize the connection, while the server socket continuously listens to the specified port for client requests. In this project, a .NET framework socket library was used.

4.3. Windows Services

Windows Services are a special type processes of the Microsoft Windows operating system. The differences between services and applications are that they (a) run in the background, (b) usually do not have a user interface or interact with the user, (c) are long-running processes till computer shuts down, and (d) automatically start when the computer is restarted. In this project, Windows service routines were implemented both client and server side (DFS). These services also have client and server socket structures to transmit data mutually.

4.4. Encryption

Cryptography is defined as a science involving advanced mathematical techniques to transmit data in a secure way that allow information to be kept secret. The data is modified using the encryption key before transmission start and the receiver converts this modified data to the original data with the appropriate decryption key. Two types of encryption methods—symmetric and asymmetric—have been successfully used for years. Asymmetric encryption is also known as public key cryptography, which is a relatively new method, uses two keys (public and private) to encrypt a plain text. The public key is used for encrypting, while the private key is used for decrypting. These two keys are created with special mathematical techniques. The public key can be known by everyone, but the secret key must be on the server-side and no one else should know it. Although the keys are different, they are mathematically related to each other. The private key is kept secret and the public key is easily shared with the target devices to which the data transfer is to be performed, because knowing this key will not contribute to solving the encrypted data. RSA (Rivest–Shamir–Adleman), DSA (Digital Signature Algorithm), Diffie–Hellman, and PGP (Pretty Good Privacy) are widely used as asymmetric key algorithms.

Symmetric cryptography—also called 'single key cryptography'—is the fastest, simplest to implement, and best-known technique that involves only one secret key to cipher and decipher

the related data. Symmetric key algorithms are primarily used for bulk encryption of blocks or data streams. There are a lot of symmetric key algorithms such as Data Encryption Standard (DES), Advanced Encryption Standard (AES), Stream Cipher Algorithm, and Blowfish. Blowfish is one of the fastest block ciphers in use and has a key length of 32 bits to 448 bits. In this study, the 128-bit key length Blowfish algorithm was preferred.

4.5. File Handling Routines

Main task of an operating system is to have a robust, fast, and efficient file system. All kinds of operating systems divide physical storage equally into 4 KB of blocks called as clusters and try to manage them most efficiently. There are basically two different disk space management concepts in use. The first one is the Unix-based bitmap method. In this method, block storage device divided into clusters and each cluster has a corresponding one bit where 1's represents free disk blocks and 0's represents allocated blocks. There are as many bits as the number of blocks. This map structure is used to find available, empty disk blocks for the files will be added to the file system. Another method for disk space allocation is one which windows based operating systems use successfully. This method is called File Allocation Table (FAT32) which uses linked list data structure to point out used or empty blocks. These two methods have different advantages and disadvantages. We use both methods to make a better hybrid solution. We use bitmap structure in the header section to see whether clusters are empty or not. To track the file chains, a linked list-based FAT structure is used; these concepts were explained in Section 4.

4.6. Programming Languages

Microsoft Visual Studio 2017 framework was used in this project. Also, C# programming languages was preferred to implement the entire system.

4.7. Brief System Overview and Integration

Medical imaging has made great progress in last few decades. Over the years, different types of medical imaging have been developed such as X-ray, computed tomography (CT), molecular imaging, and magnetic resonance imaging (MRI). HIMS use these files for reporting, data transfer, diagnostic, and treatment purposes. These images, which doctors can use to help better understand a patient's condition, are the most important part of diagnosis and treatment. In this study, client application refers to a health information management system which is running on a web-server with a static IP.

There are major theoretical and conceptual frameworks that were used in this study and can best be summarized under three headings: service-routines (client and server side), security issues (for secure connection), and developing distributed file system architecture (the main part of our system). Service routines have been implemented by using client–server socket tools to accomplish secure communication between client-side and server-side, client-service routine (CSR), and data-node service routine (DNSR). These services communicate with each other using transmission blocks that are shown in Figure 1, in JSON data structure over TCP/IP. Library files (DLL files) must be installed on the client application for the client application to integrate into the system. The CSR is responsible for sending the medical image files regardless of its size and type to DNSR and reading them on demand. The CSR sends the client application's requests (HIMS) to DNSR with a secure connection. Also, one smaller Windows service has been implemented for replica nodes to search the files. Moreover, this service provides encrypted data transfer with the other nodes.

The server side is the important part of this work, it is a comprehensive kernel service responsible for (a) listening to the specified port number for any client application request; (b) the authentication process includes checking CSR IP value; and (c) processing reading, writing, and deleting operations according requests.

Another important key part of the study is achieving a strong security level which includes: (a) primarily static IP value of CSR; (b) JSON and BSON data structure for data transfer; (c) symmetric encryption algorithm.

Thanks to security measures, the security of the developed middle layer platform has been increased dramatically.

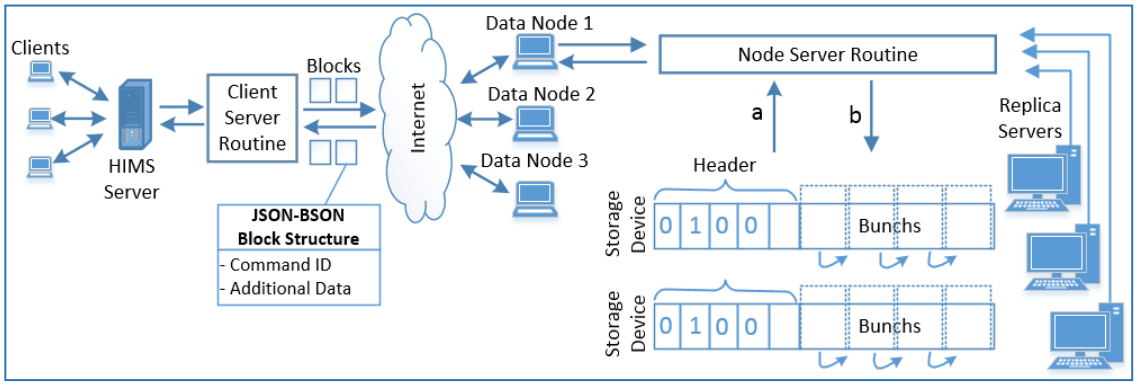

Figure 1. Windows service routines and JSON packages.

5. System Overview

In this study, a fast, secure, serverless, robust, survivable, easy to manage, and scalable distributed file system has been developed for medical big data sets. This system is a smart distributed file system developed primarily for Kırıkkale University and called Kırıkkale University Smart Distributed File System (KUSDFS). KUSDFS is structurally designed to manage large amounts of data such as Hadoop, NFS, and AFS. The contributions of this study are explained in detail below. Today, the amount of data in the health sector is near 10% of data produced globally [42]. Healthcare software systems do not tolerate waiting from server that they order a process on, so all transactions should be run as soon as possible or within an acceptable timeout period. Hadoop and the other solutions have a lot of burdens to accomplish these tasks conveniently. Because they are mainly created for big data storage needs and to overcome grid computing operations by the MapReduce framework. As a result, for Kırıkkale University healthcare software, there is a crucial need to implement its own distributed file system, here is the key factor in our study.

Comparisons with other data processing methods have been made to evaluate the performance of the system. The characteristics of the comparison systems are:

Hadoop

The Hadoop configuration used for comparison includes one name node and three data nodes. Red Hat Enterprise Linux Server 6.0 runs on each node. Also, Java-1.6.0, MongoDB-3.4.2 and Hadoop-2.7.2 is installed on each node.

CouchBase

One of the systems used to compare is Couchbase, which is a popular NoSQL database system. In a study by Elizabeth Gallagher, she claims that Couchbase is clearly as powerful as the other popular NoSQL databases [43]. In the system, each bucket is made up of 20 MB of clusters. The test machine has 6 GB of RAM and 200 GB of storage space. Microsoft SQL Server 2014 is also selected and installed on the same machine to make comparisons for RDBMS database engine.

System components are: client applications, data nodes, virtual file system, and replica nodes.

5.1. Client Applications

Client applications are remote applications that receive filing service from KUSDFS system. The applications are platform independent because they make all requests on TCP/IP. It is enough that applications needing filing service must have a static IP. Client applications can easily use this

system commonly as a 'write once read many' (WORM) system for filing and archiving purposes. An application that wants to receive filing service from KUSDFS should install the Dynamic Link Library on its system. In this system, GetAvailableDatanodeIp, ChangePassword, SaveFile, ReadFile, DeleteFile, GetFileFromReplica methods are available and detailed further on next sections. The client application communicates with the KUSDFS data node, which is reserved for itself and manages all its operations via this node. Authentication for secure logon needing username and password is done by using symmetric cryptography methods. However, static client IP is also registered on the server to strength connection security. As there is no limit to the number of client applications, these applications can use more than one data node defined on KUSDFS when needed. Here is one of the key differences of our system that others do not have. The general structure of our system is shown in Figure 2.

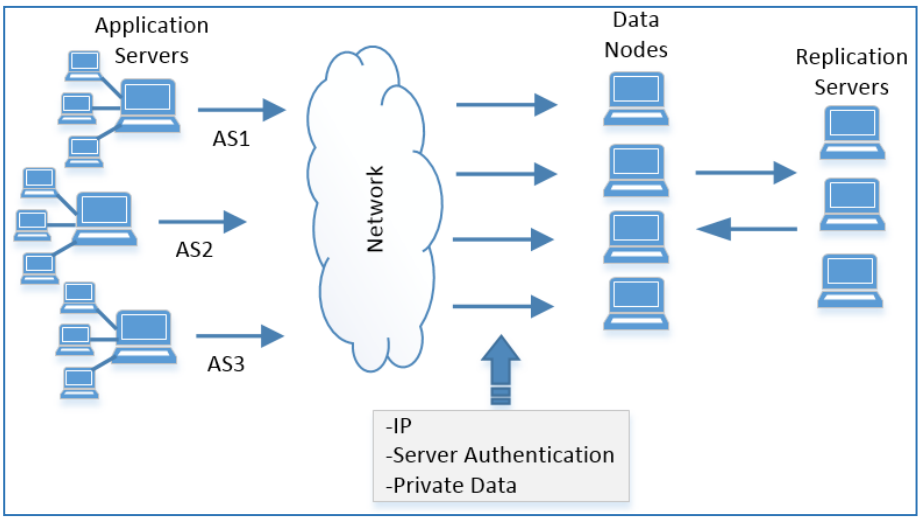

Figure 2. System overall structure.

5.2. Nodes

One of the essential elements of the system is node elements, and there are two types: data nodes and replica nodes. Distributed systems usually use head name nodes and data nodes. In this work, all the functions of the head node are inherited to the data node, so only data node is used. Because, in other systems, when head node is crushed, all the system will be unavailable usually, perhaps it causes data loss [19–32]. These nodes have quite functional properties in that they can collect data node and server node functions. In this way, the survivability and availability of the system is very good. In the whole system, even if only one node remains, the system can continue to operate. This is one of the features our system possesses.

5.3. Data Node

It is known that data nodes are responsible for storing and managing files that come from the client. The client applications communicate with data nodes via TCP/IP network service. In this work, each of the data nodes has the same priority level to process the filing services for the responsible client of that. The software, dynamic link library developed by us, running in the client provides a secure connection to data nodes via symmetric-key algorithms. However, application servers have a static IP address that are served by corresponding data nodes. Through this IP, it is connected to the data node. Data nodes also control the IP address of requests whether that is on the registration list or not. In addition, the client can encrypt the data that they want to send by the symmetric key algorithm and securely send the encrypted file to the data node optionally. Likewise, both static IP with secure authentication and encryption of files have achieved a strong security level. The data nodes will store requests from the client on the virtual file system that is a large file on the data storage device. One of

the most prominent features of the system is the developed virtual file structure. Also, each of the data nodes can serve in more than one client application. The nodes, to achieve optimal load balancing, are used for file storage in order. Also, the client application can be connected to multiple data nodes, so each node can serve multiple client applications at the same time. With this property, the system has both good scalability and availability.

5.4. System Service Architecture

There is a service running on each data node to accomplish all the functionalities that the system has. This program listens to the client applications by using the server socket structure through the port allocated to it (443 which is reserved for HTTPS, but we used this port). When the service routine is enabled:

1. It identifies all data storage devices in the computer and prepares the device list.
2. It creates a large empty file that covers 80% (optimum file size according to our experiment/test) of each element in the device list (this file has been created for each drive only once). This description is defined in Figure 3. The file corresponding to a disc drive in the computer has a disc header and array-based bunch list in which each bunch has 10 MB of size. Disc header contains a bitmap of bunch list and each bunch is represented in a header by one bit whether it belongs to a file or not.
3. The structure of this file is described below as a virtual file system.
4. SystemDataFile located on the first node of the system is read by the data node when activated. This file contains comprehensive information for all data nodes, replica nodes and client applications in the system. This size is a rather small size. As shown in Table 1, all data nodes get this list from the first node that has constant IP value when the computer restart. Then, when a new node is added to the system or there is an update in the list, the system administrator sends the CheckServerList function of the service routine to all the data nodes. Each client application is connected to the data node that is assigned to it. (At least one data node). The functions used are GetAvailableDatanodeIp, SaveFile, ReadFile, DeleteFile, CheckServerList, and GetFileFromReplica.

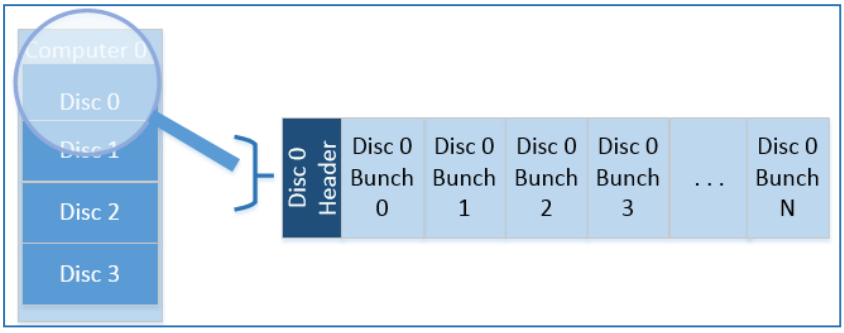

Figure 3. System data storage device architecture.

Table 1. System data file containing registered IP list.

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, also responsible for the client application, runs/acts like a load balancer. This node sends the IP address of next data node that is responsible for file storage operation. Therefore, saving file operation is done in two steps. When the client application wants to store the file, it takes the IP value of the appropriate data node from corresponding data node. As mentioned before, corresponding data node means client applications has a proxy node which is responsible for routing operation.

As shown in Figure 4, the client gets the IP address of the data node that will store the file from the proxy node before storing the file. Load balance functionality provides different data node IP for SaveFile operations. This operation is done with the GetAvailableDatanodeIP function. The system saves the client file to the data node which is nearly got from the proxy node by using SaveFile function. The SaveFile function saves the file in its virtual file system with a linked list structure and changes the corresponding bunch in bitmap (a hybrid of Unix bitmap and Windows FAT32 structure with revised version). Because of this operation, the client gets a unique file name that has the starting bunch number and the data node id of the file. In summary, when the client application wants to store the file, it gets the IP address of the data node that will store the file. In this way, the client switches to data communication with the target data node to store the file. In systems that use head node structure, the data is transferred to head node, and the head node writes this data to the data node. In this schema, the amount of data transferred is doubled. Because data first goes from client to head node and then goes to the data node. This type of work also reduces the number of concurrent connections to the head node. But in our system, the client communicates with the data node that will store the file, and the data is transferred once. In this way, the concurrent connection number is maximized.

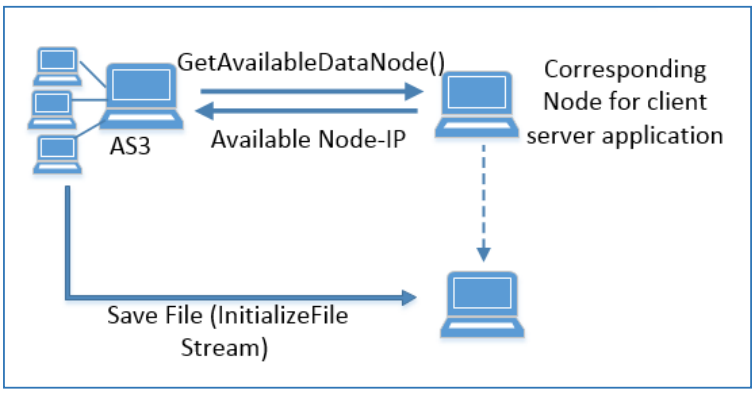

Figure 4. File storage processing.

The client receiving these values deals with only this data node in subsequent operations (in the commands ReadFile and DeleteFile) and run the requests. Only when the ReadFile operation fails, the client app gets one of the other copies of the file with the GetFileFromReplica command.

When a new client is added to the system, and change is made to the data nodes, the system administrator sends CheckServerList command to the all nodes.

5.6. Virtual File System

The system creates a file that occupies 80% of the total free space on the physical device. At this point, as shown in Figure 3, the file will form a continuous area on the physical drive. Thus, the required disk I/O will decrease. This file is divided into blocks called a bunch of 10 MB Figure 5 shows a bunch structure. A bunch consists of 3 bytes of next bunch information, 1 byte of data indicating to which replica server is kept, 4 bytes of client application IP, and 50 bytes of the file name and data block. The bunches behave exactly like an array, and the index number starts from zero. As shown in Figure 6, the disc header consists of 3 bytes of data holding the amount of bunch and 200 KB of data in bitmap

format. Whether each bunch is empty or not is controlled by the bitmap structure, in which each bunch is represented by one bit and '1' represents used or reserved bunch otherwise the bunch is empty, which means it is ready to be used in the file chain. This bitmap field size is 200 KB. Thus, a storage area of 16 TB is reached. Each bunch holds a 3-byte pointer that holds the next bunch in itself. Files that are bigger than a bunch size are kept in the file chain. As shown Figure 7, each bunch points to the next data block with of 3 bytes next pointer. The pointer holds the index of the next bunch. If this pointer is '0', it means that the bunch represents EOF, in other words, this bunch is the last one of the file chain. Data nodes are responsible for storing, reading, deleting, and backing up the files sent to them. One of the most basic features of this structure is that the starting bunch number of the file stored in the nodes is sent with the file name and in this way/thus the reading starts with one disk access. With this feature, system performance is quite satisfactory. 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.

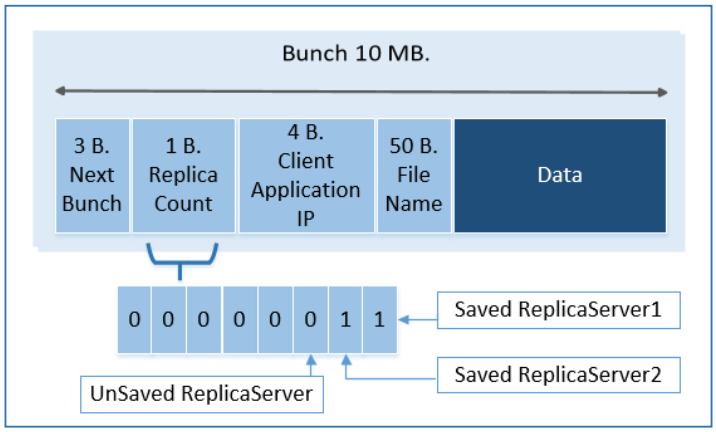

Figure 5. Bunch structure of file system.

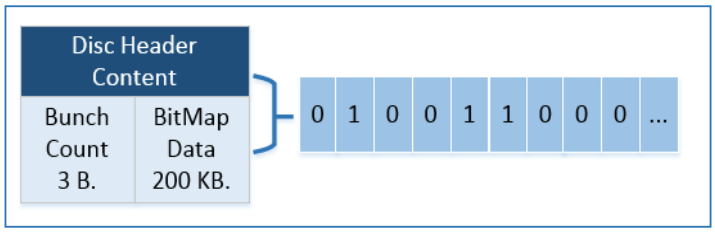

Figure 6. Structure of disc header.

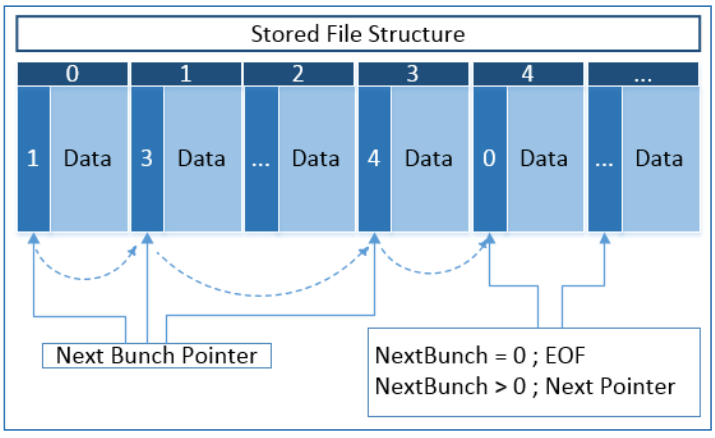

Figure 7. Structure of file data bunch.

Table 2. The response times given by the systems for different file size (ms.)

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

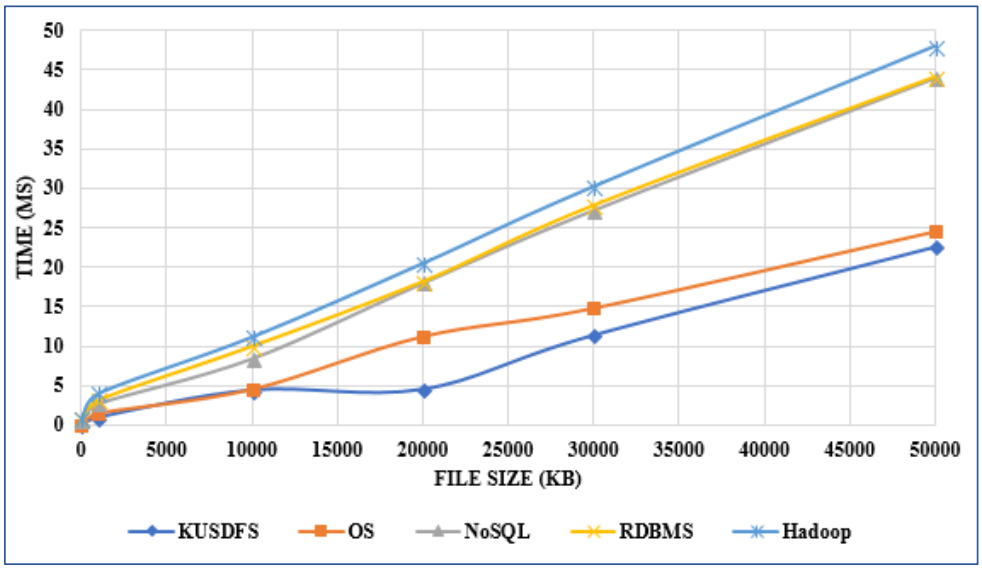

Figure 8. The response times given by the systems for different file size (ms.).

Firstly, the developed system can be used by more than one client application server. The client application server is matched to one of the data nodes, and all requests are forwarded to this server (data node). A data node can serve multiple client servers; this increases the survivability of the system. Application servers may communicate with different data nodes, and this definitely will not affect the operation of their systems. Each of the data nodes in the system can serve multiple client servers at the same time. In short, all nodes of the system can be used as a server when requested. With this feature, survivability, availability, and reliability values give very good results.

The client application works with the data node to which it is connected, when the client-server wants to store the file, requests the address of the next data node from data node. The client-server that receives this address writes the file to the data node at this IP address. The application server writes its files in order to the data node in the system. The only task of the data node that the application is associated with is to give it an IP address. In fact, this structure resembles the name node in the Hadoop architecture.

5.7. Replica Node

Replica nodes, which are used to increase the system's fault tolerance in the DFS as an indispensable backup method, are also used and are much easier to process on this system. First, these nodes are used to make copies of files that are stored by data nodes. The data nodes asynchronously send the uploaded files to replica nodes. Replica nodes store these files in the underlying operating system file structure. Here, no special file structure or any other file processing strategies are used, but the operating system's filing service is done. This means replica nodes run as known file servers that are only responsible for storing files on the operating system volume-directory structure. The replica count was set at three as the default replica count for DFS, for all nodes that replica count can be raised to eight.

All replica nodes have a service responsible for storing, deleting, updating, forwarding, and searching the files in the system. So, when a file search arises or is needed in one of data nodes or client, SearchFile command is sent to all replica nodes, then, at the same time replica nodes starts to search targeted file in their volume structure as grid computing, at the end of concurrent search process, each node returns the search results indicating that search process is successful or not. In other words, when a file wants to be searched by a node, SearchFile messages are sent to all replicas, replicas work in parallel, and the result is transmitted to the data node requesting the file. Replica nodes not only make file searches but they also process ReplicaWrite, ReplicaRead, and ReplicaDelete commands which denote the operations applied to a file in writing, reading, deleting, and updating on the replica node, respectively.

5.8. Functional Features

1. Serverless architecture as preferred instead of using name node : The term "serverless" does not mean that servers are not used, or there is no server. This only means that we no longer have to think too much about them [44]. At the same time, the data node at the same level are considered serverless hierarchically. Each of the four nodes used in this study serves file storing operations for different types of client applications. When any of these nodes is disabled, the system will automatically route the client to the other data nodes. In this way;
    - The “serverless” schema does not mean servers are no longer involved. It simply means that developers no longer should think that much about them. Computing resources get used as services without having to operate around physical capacities or limits.
    - It is possible to use different sources (more than one hard disk) on the same machine.
2. Corporations need small and medium block sizes: Hadoop and the others are generally configured for at least a 64 MB block size, this approach makes it difficult to acquire an excellent choice for small and medium-sized big data, this phenomenon results in as those not available for small and medium size big data problems.
3. Name node crash recovery problem is solved. The fault tolerance of the system is very high because there is no name node.
4. An easy to manage virtual file system has been produced on the underlying operating system. A mixture of both bitmap and linked list structure was used for managing the bunches which have a fixed size of 10 MB.
5. The start address of the file is saved in client applications that give us sufficient performance gain.
6. The most important era of our system is any of the data nodes may be a server for any application. In other words, any machine in the system can be a file server.
7. All the IP lists which contain data nodes and replica IPs and secure connection IPs are available for all nodes in the cluster.

Non-functional specifications that KUSDFS have:

1. Performance (reading): when we compare data processing speed with other systems, we have achieved very good performance.
2. Scalability: due to the features it has, it is sufficient to install only the service program to add new nodes into the system. In this way, the system is designed to be able to serve thousands of nodes, while it can work with several nodes according to the institutional needs.
3. Survivability: survivability is that the system survives despite all the negativities and maintains its minimum functions. In the developed system, even if at least one of the nodes is active, the system is still able to work.
4. Availability: availability is that the system delivers its services successfully every time. This has been done with the developed system. Especially since there is no name node, the survival of the system is quite high.

5. Security: static IP and synchronous encryption method were used to strengthen the security level of the system.
6. Minimum cost: data nodes used in the system are ordinary machines and do not have an extra feature.

6. Discussion

Big data has long been an issue of great interest in a wide range of fields, especially in healthcare. Recently, researchers have shown an increased interest in managing this problem by using the Hadoop ecosystem. However, much of the research up to now has been descriptive in nature of how data analytics methods or MapReduce tasks are used on the data, amd they have not addressed what will be done for efficient data storage with minimal internal fragmentation for small, medium, and big institution's needs. This study seeks to obtain a successful solution which will help to address these gaps.

It is hoped that this research will contribute to a deeper understanding of problem-oriented stand-alone solutions.

The distinctive strengths of this study are:

1. Read–write performance due to the hybrid architecture.
2. Robustness and availability with the help of using no name node and being each node like a server.
3. Optimal load balancing.
4. Successful integration with cheap ordinary hardware elements.
5. Secure connection by using 128-bit symmetric cryptology.
6. Easy scalability by simply installing the library files to the node.
7. Suitable for healthcare institutions by using 10 MB block size. Figure 9 shows the strengths, weaknesses, and opportunity features of the system.

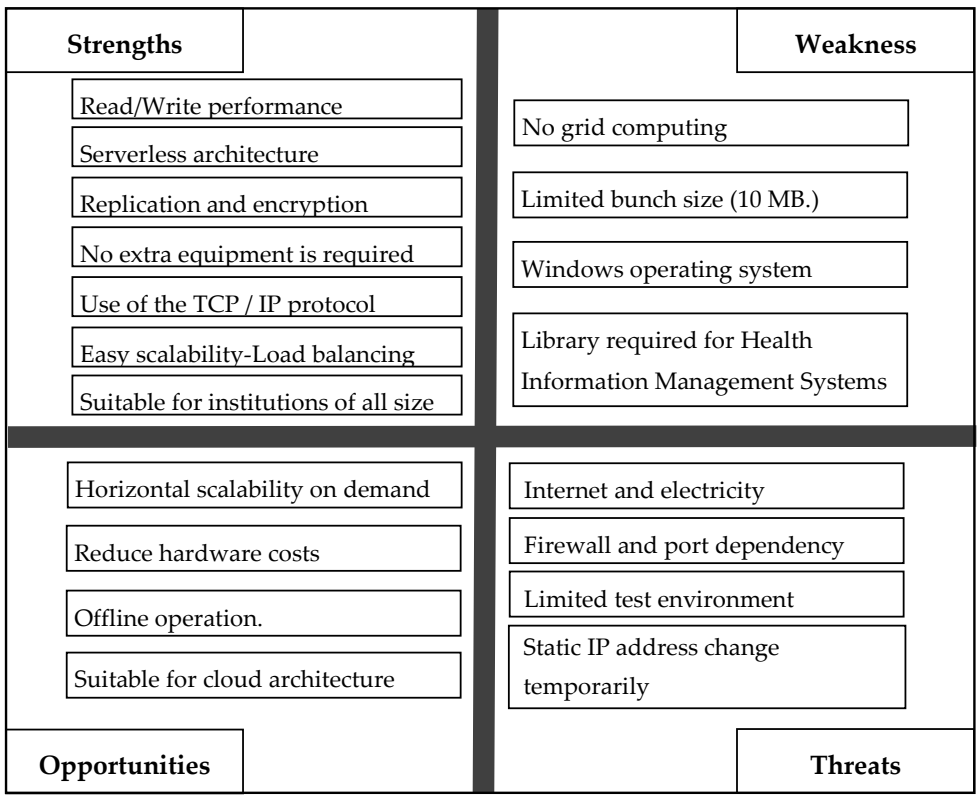

Figure 9. SWOT Analysis of the proposed system.

The limitations of the present study naturally include:

1. No grid computing facilities.
2. Fixed bunch size of 10 MB.
3. Static IP address change which causes temporary system stops, but this situation applies only to the client application, so it does not affect the availability of the system.
4. Limited test environment with four nodes.
5. Symmetric cryptography operates with a single key, and it is important to keep the key secret for the security of this cryptography.

As a result, this study has advantages and disadvantages as discussed here. It will be important that future research be conducted to improve upon this system's drawbacks.

7. 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.

System security has an acceptable security level as other distributed file systems. This is provided in two ways. The first is to check the IPs of the client machine that are served from the data node and the second is to encrypt the data that the application software sends to the data nodes.

Because a data node can serve more than one client in our system, it has a better load balance performance than that of other distributed file systems. In the same way, a client can upload data to more than one data node.

In the designed system, a disc consists of data sets called the 'bunch' and 'disc header'. The disc header holds the number of bunches found in that disc and whether the bunch is empty with the bitmap structure. A bunch holds the next bunch, the replicas of the data, the IP of the client application that loads the data, and the data itself. Using the designed file system, every bunch can point to the next bunch which is a member of the file chain.

The replication operation uses the operating system file operations system calls (in Windows called API). Data nodes send the files uploaded to itself asynchronously 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.

In the future, healthcare institutions in Kırıkkale province can be integrated to KUSDFS. At this point, the cost, workload, and technology needs of the institutions will be minimized. Instead of establishing their own archiving systems, it is possible to access and integrate with the system by only installing library files. The developed system has a fixed bunch size of 10 MB which is specialized for healthcare. To embrace dissimilar applications that have different file size and types, a dynamic bunch size might be developed in the future work. Grid computing functionalities could also be implemented to make the system more powerful and more efficient in terms of computational speed. This is an important issue for future research. In addition, GPUs—which have been more widely used in recent years—can be included in the system to achieve a more specific distributed computing platform and to accelerate grid performance. However, further work with more focus on this topic is required to establish this property.

Author Contributions: A.E. conceived and structured the system. M.Ü. performed literature search, helped in experiments and analyzed the test data. Both, authors implemented the system, wrote the paper, read and approved the submitted version of the manuscript.

Acknowledgments: This work has been partly supported by the Kırıkkale University Department of Scientific Research Projects (2016/107–2017/084).

Conflicts of Interest: The authors declare that there is no conflict of interest.

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/scaling-the-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. *Sağlık İstatistikleri Yıllığı 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-dezavantajlari-hakkinda-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.; Korukoğlu, S. Modern Dağıtık Dosya Sistemlerinin Yapısal Karşılaştırılması. In Proceedings of the Akademik Bilişim'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 Dağıtık 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. 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.
36. 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.
37. 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.
38. Muja, M.; Lowe, D. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *36*, 2227–2240. [CrossRef] [PubMed]
39. 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.
40. 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]
41. 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.
42. Hossaina, M.S.; Muhammad, G. Cloud-assisted Industrial Internet of Things (IIoT)—Enabled. *Comput. Netw.* **2016**, *101*, 192–202. [CrossRef]

43. Chandra, D.G. BASE analysis of NoSQL database. *Future Gener. Comput. Syst.* **2015**, *52*, 13–21. [CrossRef]
44. Redwrite. Available online: http://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless/ (accessed on 15 June 2017).