Universidade de Aveiro
2023

JOÃO PEDRO DOS SANTOS RODRIGUES

BASE DE DADOS ESCALÁVEL E DISTRIBUÍDA GEOGRAFICAMENTE PARA IMAGENS MÉDICAS

A SCALABLE AND GEOGRAPHICALLY DISTRIBUTED DATABASE FOR MEDICAL IMAGING

Universidade de Aveiro
2023

JOÃO PEDRO DOS SANTOS RODRIGUES

BASE DE DADOS ESCALÁVEL E DISTRIBUÍDA GEOGRAFICAMENTE PARA IMAGENS MÉDICAS

A SCALABLE AND GEOGRAPHICALLY DISTRIBUTED DATABASE FOR MEDICAL IMAGING

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática , realizada sob a orientação científica do Doutor Carlos Manuel Azevedo Costa, Professor associado c/ agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Luís António Bastião Silva, Professor auxiliar convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Palavras Chave

Apache Cassandra, Big Data, DICOM, Escalabilidade, Escalabilidade Horizontal, Imagiologia Médica, Modelo de Dados, NoSQL, PACS

Resumo

No atual cenário de imagiologia médica, o crescente volume e complexidade dos dados de imagens médica apresentam desafios significativos no seu armazenamento, recuperação e análise. Esta tese aborda esses desafios explorando o potencial de base de dados NoSQL, com um foco específico no Apache Cassandra como infraestrutura de armazenamento subjacente para um Sistema de Arquivamento e Comunicação de Imagens (PACS). Este documento começa por examinar as características distintivas do padrão DICOM para dados de imagiologia médica, a gama de opções de base de dados disponíveis, bem como as arquiteturas PACS existentes. Esta tese concentra-se no design e implementação de um sistema PACS que aproveita as vantagens inerentes do Cassandra, como a distribuição, escalabilidade e tolerância a falhas, garantindo ao mesmo tempo a integridade dos dados ao nível da base de dados. A solução desenvolvida melhora significativamente a escalabilidade, a distribuição e a tolerância a falhas, sendo capaz de conter todas as informações necessárias para reconstituir ficheiros DICOM sem depender de um sistema de ficheiros externo. Os principais objetivos deste trabalho são criar uma solução altamente escalável, distribuída, tolerante a falhas e eficiente. A solução não apenas atende a esses objetivos, mas também oferece melhorias específicas, incluindo a capacidade de reconstruir ficheiros DICOM de forma contínua sem a necessidade de um sistema de ficheiros auxiliar.

Keywords

Apache Cassandra, Big Data, Data Models, DICOM, Horizontal Scaling, Medical Imaging, NoSQL, PACS, Scalability

Abstract

In the current medical imaging landscape, the ever increasing volume and complexity of medical imaging data presents significant challenges in the storage, retrieval, and analysis. This thesis addresses these challenges by exploring the potential of a NoSQL database, with a specific focus on Apache Cassandra as the underlying storage infrastructure for a Picture Archiving and Communication System (PACS). This document start by examining the distinctive characteristics of medical imaging data,in accordance with the DICOM standard, the spectrum of available databases options, as well as existing PACS architectures. This thesis focuses on the design and implementation of a PACS system that leverages Cassandra's inherent strengths, such as its distribution, scalability, and fault-tolerance, while ensuring data integrity at the database level. The developed solution significantly enhances scalability, distribution, and fault tolerance, and can hold all requisite information to reconstitute DICOM files without relying on an external file system. The key objectives of this work are to create a solution that is highly scalable, distributed, fault-tolerant, and performant. The solution not only meet these objectives but also offers specific improvements, including the ability to seamlessly reconstruct DICOM files without the need for an auxiliary file system.

Contents

Contents **i**

List of Figures **iii**

List of Tables **v**

Glossary **vii**

1 Introduction **1**

1.1 Motivation . . . 1

1.1.1 Big Data problem in medical imaging . . . 1

1.2 Objectives . . . 3

1.3 Document Structure . . . 3

2 State of Art **5**

2.1 Medical Imaging . . . 5

2.2 DICOM Standard . . . 5

2.2.1 Digital Image Communication in Medicine (DICOM) File Format . . . 5

2.2.2 DICOM Services . . . 9

2.2.3 DICOMweb . . . 12

2.3 Relational Databases . . . 12

2.4 Non-Relational Databases . . . 13

2.5 PACS . . . 15

2.5.1 Storing pixel data in a Picture Archiving and Communication System (PACS) 18

2.6 Related Work . . . 18

3 Architecture Proposal **21**

3.1 Requirements . . . 21

3.1.1 Functional requirements . . . 21

3.1.2 Non-Functional Requirements . . . 21

3.2 Architecture . . . 22

3.3 Database Approach 23
3.4 System Configuration 23
3.5 Cassandra-based PACS 24
3.5.1 DICOM Format and Cassandra Schema 24
3.5.2 Cassandra Tables and Prepared Statements 24
3.5.3 Cassandra Secondary Indexes 24
3.5.4 Data Storage Process 25
3.5.5 Storage Process 26
3.5.6 Retrieval Process 26

4 Experiments and Results **29**
4.1 Testing scenario 29
4.2 Results 29
4.2.1 Fault tolerance 30
4.2.2 Storage Process 30
4.2.3 Retrieval Process 32

5 Conclusion **37**

References **39**

List of Figures

2.1 DICOM File Structure [4] 6

2.2 Service-Object Pair (SOP) Class [6] 7

2.3 Tag, Length, Value (TLV) Strucutre [8] 8

2.4 Hierarchical Data Levels of DICOM 9

2.5 Sequence diagram of DICOM message service element (DIMSE) C-Store Operation [9] . . 10

2.6 Sequence diagram of Query/Retrieve Service Operations [10] 11

2.7 PACS Architecture Example [20] 16

3.1 System Architecture 22

3.2 Cassandra Schema 25

4.1 Storage time for different chunk sizes 30

4.2 Storage time across frame count and file size 31

4.3 Retrieval time at image level for different chunk sizes 32

4.4 Retrieval time at the different data levels 33

4.5 Retrieval time across frame count and file size 34

4.6 Retrieval time at different data levels 35

List of Tables

2.1 Description of Services and Associated Commands 10

3.1 Prepared Statements 26
3.2 Structure of the Retrieval CQL Commands 28

Glossary

3D	three-dimensional
ACR	American College of Radiology
ACID	Atomicity, Consistency, Isolation, and Durability
AE	Association Entity
BASE	Basically Available, Soft state and Eventual consistency
CQL	Cassandra Query Language
CR	Computed Radiography
CRUD	Create, Retrieve, Update and Delete
DICOM	Digital Image Communication in Medicine
DICOMweb	DICOM Standard for web-based medical imaging
DIMSE	DICOM message service element
GB	Gigabytes
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IOD	Information Object Definition
IP	Internet Protocol
JSON	JavaScript Object Notation
LTS	Latest Stable Version
MB	Megabytes
MR	Magnetic Resonance
NEMA	National Electrical Manufacturers Association
NoSQL	Non Structured Query Language, also known as Not only SQL
P2P	Peer to Peer
PACS	Picture Archiving and Communication System
QIDO	Query-based on ID for DICOM Objects
RAM	Random Access Memory
REST	Representational state transfer
RPC	Remote Procedure Call
RDBMS	Relational Database Management System
RQ	Request
RSP	Response
RS	RESTful Services
SCP	Service Class Provider
SCU	Service Class User
SM	Slide Microscopy
SOP	Service-Object Pair
SQL	Structured Query Language
STOW	Store Over the Web
TCP	Transmission Control Protocol

TLV	Tag, Length, Value
UID	Unique Identifier
ULP	Upper Layer Protocol
UTF	Unicode Transformation Format
VR	Value Representation
WADO	Web Access to DICOM Persistent
XML	Extensible Markup Language

CHAPTER 1

Introduction

1.1 Motivation

Medical imaging has become one of the most useful tools in diagnosis. The systems that allow their efficient storage and retrieval are of the utmost importance. Fault-tolerance and high availability are also required for any good medical imaging storage solution. With the ever increasing ease of generating medical images and with the growth in modalities and the corresponding file size increase, medical imaging has become a Big Data problem. Traditional database solutions become a expensive proposition with the difficulties associated with scaling these type of systems to that volume of data. The medical image scenario also has been changing alongside the Web 2.0 revolution, which open the possibility of sharing medical information across distinct geographic locations and across different institutions, to more easily adapt to this new scenario the storage solution needs to be easily distributed.

1.1.1 Big Data problem in medical imaging

Big Data can be defined as data set of massive proportions and high complexity that are difficult to store, analyze and access for further processes. Big Data is a complex problem to address with the three main issues being velocity, volume and variety. Big Data tends to be generated from a variety of sources, and might range from structured, semi-structured or unstructured data. Velocity comes into play not only in accessing the data but making it available to the processes that are to be run using said data [1].

Medical imaging is one of the poster cases of Big Data, with the industry moving from films to digital imaging the volume of data to be stored has been growing exponentially. This increase in volume is the result of development of medical imaging technologies such as three-dimensional (3D) imaging, the rise of mobile applications in the medical imaging context, new capturing devices and sensors, and the development of PACS systems. Medical images also presents with the problem of great variety of data. The data can vary, for instance, with the type of modality, in which case the generated DICOM files may have different fields and information depending on the modality. Medical imaging data is being generated in real-time and at a high rate, with the referenced above developments and the growing adoption of electronic health record technology, this rate is only expected to increase. The access to this data also needs to be pretty efficient to not negatively impact the physicians workflow. So medical imaging has all the hallmarks of a Big Data problem.

Due to the rate and the size of the generated medical images, the data archive must be easily scalable to meet the growing volume of data. Traditional database or Relational Database Management System (RDBMS) scale vertically, which means that to increase performance the machines running the database need to be replaced with machines with higher specifications (more memory and higher processing power) to meet the new requirements, at some point there are cost and technological limitations. Non Structured Query Language, also known as Not only SQL (NoSQL), databases were developed with Big Data in mind, so they tend to allow for horizontal scalability, to increase performance more machines are added to database management system, which results in lower cost when upgrades are necessary. It is clear that NoSQL technologies are probably the answer to Big Data problem in medical imaging.
With the growing adoption of electronic health record technology, the reduction of cost of medical imaging equipment and the outsourcing of PACS infrastructure to data centers, the ease of distribution of this records is becoming a growing need. NoSQL solutions in this aspect edge out traditional databases, since the process of distributing information in the latter tends to be a complex and time consuming process, while some NoSQL were developed with a distributed nature as a core feature, to take advantage of the rise of cloud based services [2].

1.2 Objectives

Based on the issues discussed above, some aspects of the developed solution became clear such as the following:

- Scales horizontally
- Easily distributed across multiple regions and institution
- Fault-tolerant with no single point of failure
- Performant

One of the proposed goals is to endow the database with the capacity to store the full pixel data of the medical images that are stored.

1.3 Document Structure

This section provides an overview of the organization and structure of this thesis. It outlines the different chapters and sections that will be presented. The document structure is designed to enhance clarity, coherence, and logical flow, enabling readers to navigate through the thesis smoothly.

Chapter 2: State of the Art

In this chapter, a comprehensive review of the existing relevant studies related to the research topic is presented. It details information about medical imaging current scenario, the DICOM standard and the associated services, database concepts and PACS that is pertinent to the current research, establishing the theoretical framework upon which this study is based.

Chapter 3: Development

This chapter describes the system requirements and architecture, explains why the selected database technology was chosen, the settings options and thePACS solution developed.

Chapter 4: Results and Analysis

It outlines the data sources, data treatment and sampling methods used in gathering and treating time metrics to better understand the behaviour and usability of the solution. These metrics are represented in graph form to enhance the understanding of the findings.

Chapter 5: Discussion and Conclusion

This chapter provides a discussion of the performance of the developed system in light of the existing solutions. An analysis of the limitations and significance of the results is made, and offers interpretations and explanations for the findings. Finally, the chapter includes suggestions for further improvements and additional functionalities to be implemented.

By structuring the thesis in this manner, the reader will be able to follow a logical progression of the research, from the introduction to the conclusion.

CHAPTER 2

State of Art

In this chapter, important concepts regarding medical imaging, its information standards and types of databases will be presented. In this chapter will also be address the concept of PACS and their conventional architecture. A specific PACS system, Dicoogle, will also be address, since one of the original objectives were to add functionality to this platform.

2.1 Medical Imaging

According to Anke Meyer-Baese and Volker Schmid, medical imaging can be defined as "Medical imaging deals with the interaction of all forms of radiation with tissue and the design of technical systems to extract clinically relevant information, which is then represented in image format." [3] With the technological revolution experienced in this area, medical imaging and their storage solutions have became of the utmost importance.

2.2 DICOM Standard

DICOM came to exist due to the need to standardize the communication and information formats used in medical imaging. Before DICOM different manufactures had different proprietary communication protocols making communication between machines of different manufactures unfeasible. To this effect National Electrical Manufacturers Association (NEMA) and American College of Radiology (ACR) came together to form a committee with the goal of creating a standard that satisfied the combined requirements of radiologists, physicists and equipment vendors. This standard eventually came to be called DICOM, suffering changes throughout the years to better meets the growing requirements. With standardization communication between machines of different manufactures became easier, allowing for a better information flow between different medical institutions.

2.2.1 DICOM File Format

The DICOM file format is divided in a file header and a data set, with the header containing a 128 bytes preamble followed by a 4 bytes DICOM prefix. This preamble may be included or not, and it is used to facilitate access to the images and other data contained in the DICOM file by defining which commonly used image file formats the file is compatible with. When this preamble is not provided all of its bytes are set as 00H and the file is accessed using a DICOM application.

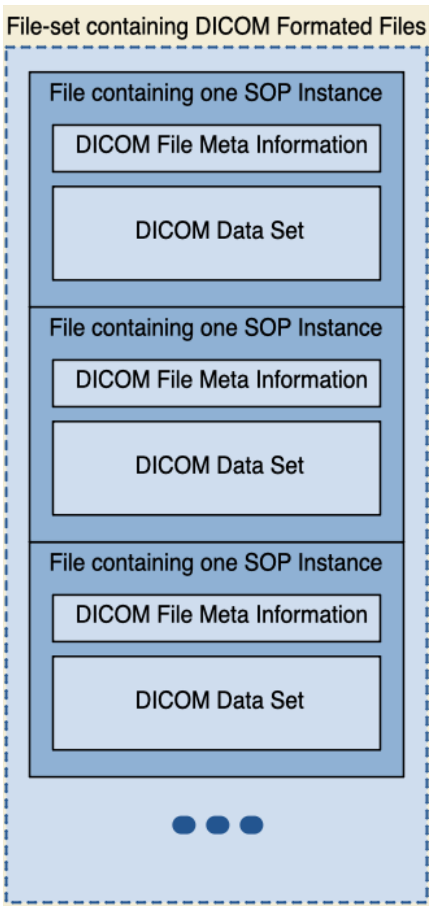

File-set containing DICOM Formated Files
File containing one SOP Instance
DICOM File Meta Information
DICOM Data Set
File containing one SOP Instance
DICOM File Meta Information
DICOM Data Set
File containing one SOP Instance
DICOM File Meta Information
DICOM Data Set

Figure 2.1: DICOM File Structure [4]

Each file normally only has one data set that represents a single SOP Instance related to a single SOP Class (and the corresponding Information Object Definition (IOD)), some specific IOD are defined to allow multiple frames inside the same file. A SOP class is the conjugation of a DIMSE and an IOD. The SOP class definition contain semantics and rules that can restrict the use of services in DIMSE service group and the attributes in an IOD. IOD is defined by NEMA as "an object-oriented abstract data model used to specify information about Real-World Objects"[5], meaning that an IOD specifies the shared proprieties of a class of Real-World objects. IOD serves to provide a common view of the information to be exchanged by the different communicating Application Entities.

The data set also includes the Transfer Syntax Unique Identifier (UID) which represent the Transfer Syntax used to encode the DICOM File Meta Information. Data Sets are constituted of a varying number of Data Elements.

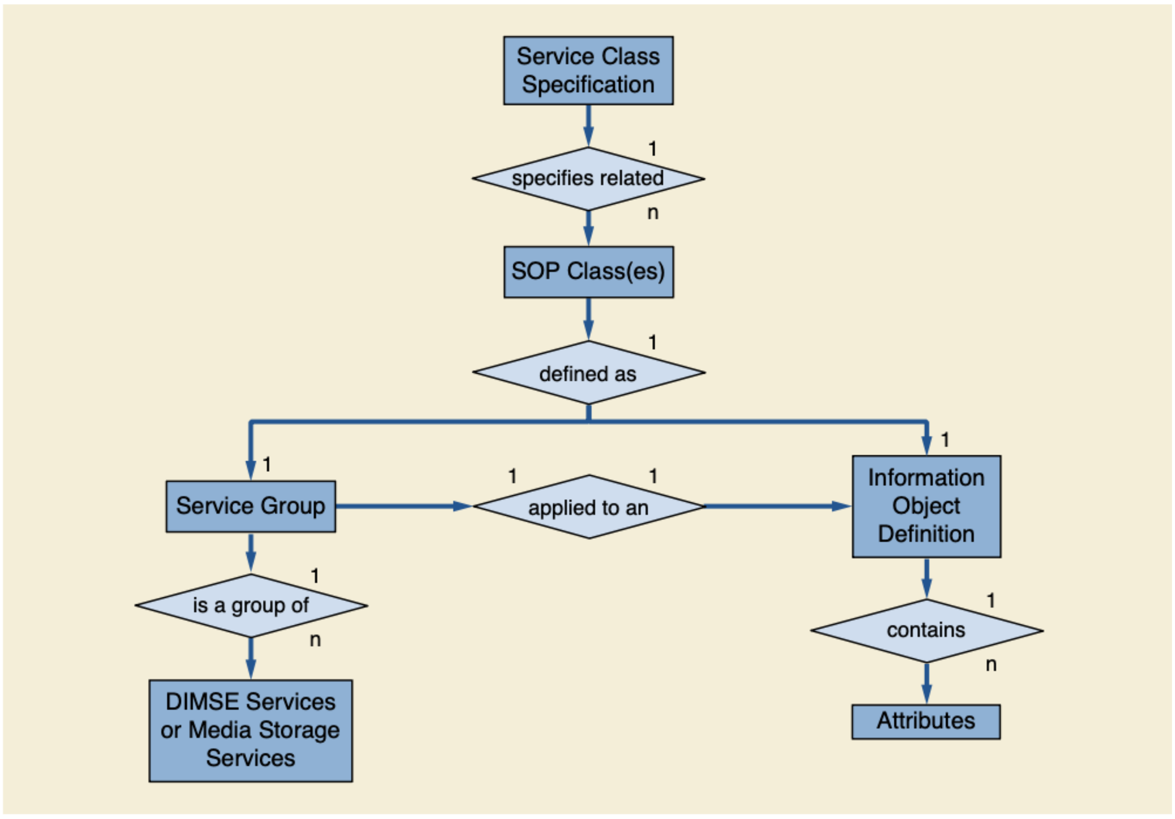


Figure 2.2: SOP Class [6]

TLV structure

Data Elements are encoded following a TLV structure, containing the encoded values of that object attributes. The content and semantics of these attributes depend on the type of IOD. The different formats of describing a image (Pixel Data, Curves, Mesh, ...) are data elements whose interpretation depends on other related elements [7]. Data elements are uniquely identified by their Data Element Tag (expect in some cases in a nested Data Set where it can be repeated), the Data Elements are ordered in ascending order of the Data Element Tag value. Data Elements come in two different types : standard and private. Standard Data Elements have an even Group Number, while Private Data Elements are odd, some of this group number cannot be utilized since their are reserved for DIMSE Commands or DICOM File Formats. Data Element can be structure in one of three ways, with two of them containing the Value Representation (VR) of the Data Element (Explicit VR) differing in the way their lengths are expressed, while the other does not contain the VR (Implicit VR). Whether the Data Elements are explicit or implicit VR is determined by the negotiated Transfer Syntax, with the type of VR being consistent inside a Data Set whether it is nested or not. As referred before Data Element is structured in a TLV format, with the Tag being a 16-bits unsigned integer functioning as an unique identifier (Group and Element Number) of the element, Length defines the size in bytes of the value field and Value is composed of binary data representing the value of Data Element. If the negotiated Transfer Syntax defines an explicit VR, between Tag and Length there is an additional field VR that defines the data type of the element, in the case of implicit VR the information of the type of data is only present in the Tag value.

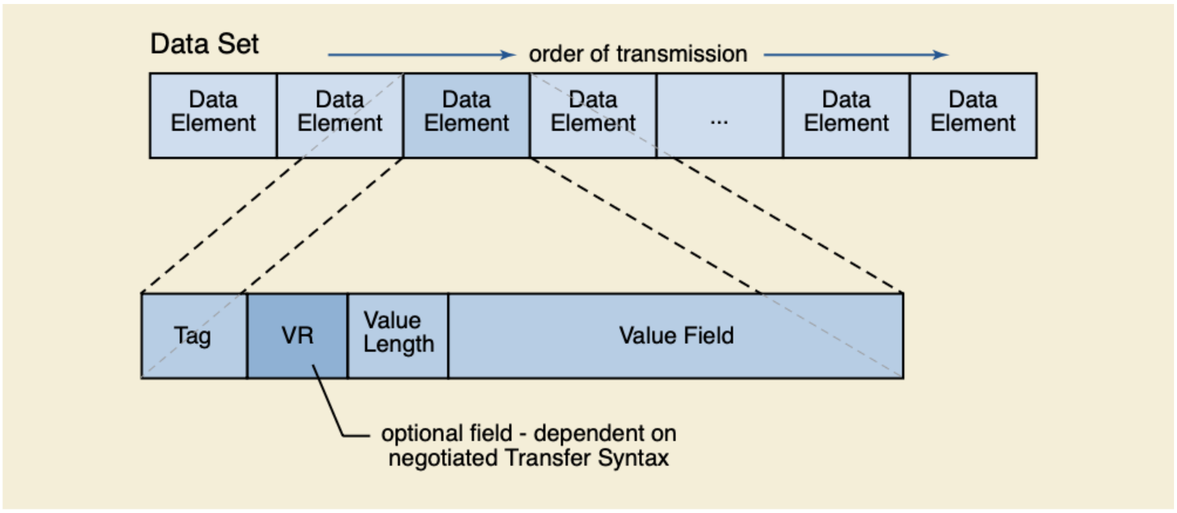


Figure 2.3: TLV Strucutre [8]

Hierarchical Data Level of DICOM

DICOM objects are designed to be identifiable by the embedded information present in the header, with this information being organized in an hierarchical fashion that models real world use cases. This hierarchy can be divided into four different levels: a patient level that identifies the person receiving the imaging procedure; a study level that identifies the imaging procedure; a series level since a study maybe be composed of multiple series which can represent different modalities, multiple scans inside the same modality or a single scan with the information being reconstructed in different ways; and finally an image level that identifies each image uniquely. Each of this level object are uniquely identified using a UID. A DICOM UID is composed of two parts: a root and suffix. With the root portion uniquely identifying the organization where the DICOM object was created and the suffix portion identifying uniquely the specific object inside the scope of the organization.

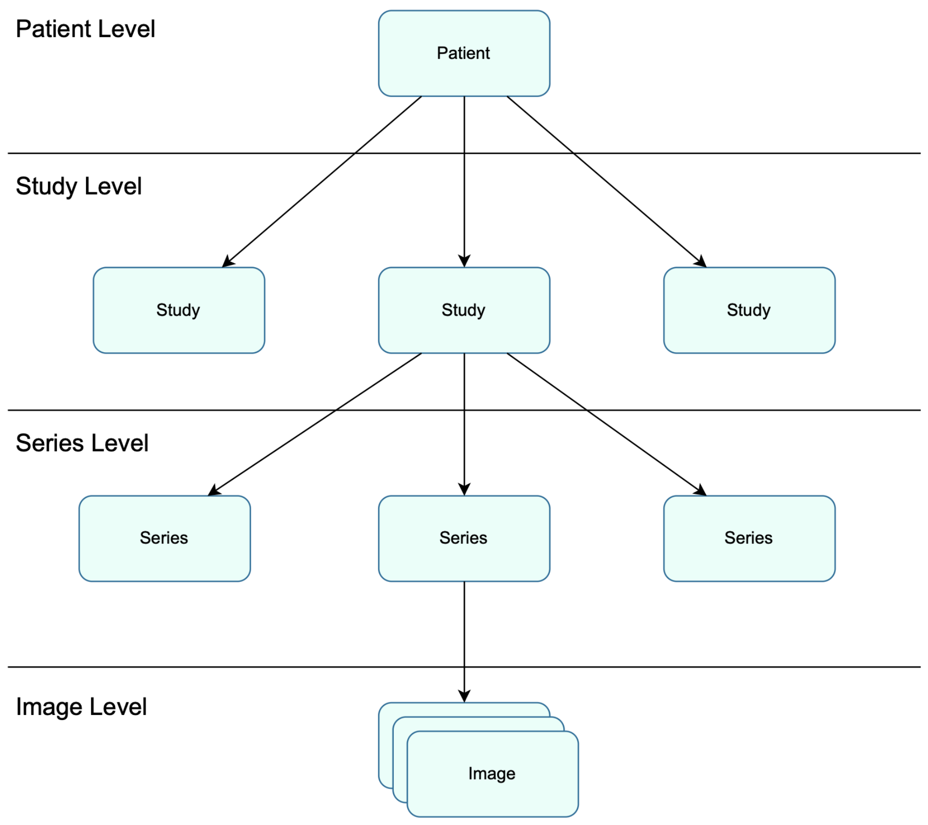


Figure 2.4: Hierarchical Data Levels of DICOM

2.2.2 DICOM Services

DICOM did not only standardize the format to represent medical information, but also the communication between all involved systems in medical imaging acquisition, storage and retrieval. DICOM defines a Upper Layer Protocol (ULP) with the underlying protocol used for communication between medical devices working over Transmission Control Protocol (TCP)/Internet Protocol (IP) with the encapsulated information structured in a TLV fashion. This communication is based on services specified by DICOM standard, each service supposes the communication between a Service Class Provider (SCP) and a Service Class User (SCU). The SCP is the DICOM node who provides the service, works as a server in a network point of view, and the SCU is a DICOM node who uses or consumes the service provided, works as a client from a network point of view. For example a PACS would typical be the SCP, while workstations or modalities would function as a SCU. Each DICOM device can be identified uniquely inside a network by the combination of its Association Entity Title, a IP address and a port number, which is used to establish communication inside said network. For the communication between two DICOM entities to be successful both entities must support the same service and the same type of object, with the two entities supporting different roles (one functioning as a SCP with the other being the SCU). This combination is called a SOP Class. So the first step in establish a connection between two Association Entity (AE) is the association stage, where both entities negotiate three key negotiation parameters, Application Context, Presentation Context, and the User Information Items, that will define if an association can be establish and determine if the other services can be achieve by this specific entities. Services standardized by DICOM might be composed of multiple commands. The following table shows the most common services and their DICOM commands [6].

Services	**Associated Commands**	**Service Description**
Storage	C-STORE	Stores a DICOM file into a PACS
Query/Retrieve	C-FIND C-MOVE/C-GET	Search and return information or images from a PACS
Worklist Management	C-FIND	Returns the requested worklist

Table 2.1: Description of Services and Associated Commands

The most pertinent services for this thesis will be explained in more depth, since the other services will not significantly impact the performance of the work being developed.

- **Storage Service:** It transfers DICOM structured information from one DICOM node to another. This service follows the regular DICOM pattern, with a DICOM node functioning as a server (i.e.DICOM Storage SCP) for requests made by other DICOM nodes acting as clients, (i.e. DICOM Storage SCU). The SCP waits for a SCU to establish a connection followed by a storage request; after the request has been processed, the SCP sends a storage response to the SCU. At a protocol level, the service is implemented using the C-STORE messages: after the SCU establishes a connection with a SCP, a C-STORE-RQ (Request) is sent; the message also contains the dataset to be transferred, the SCU then waits for C-STORE-RSP (Response) sent by the SCP, the response communicates if the storage request was successful or not.

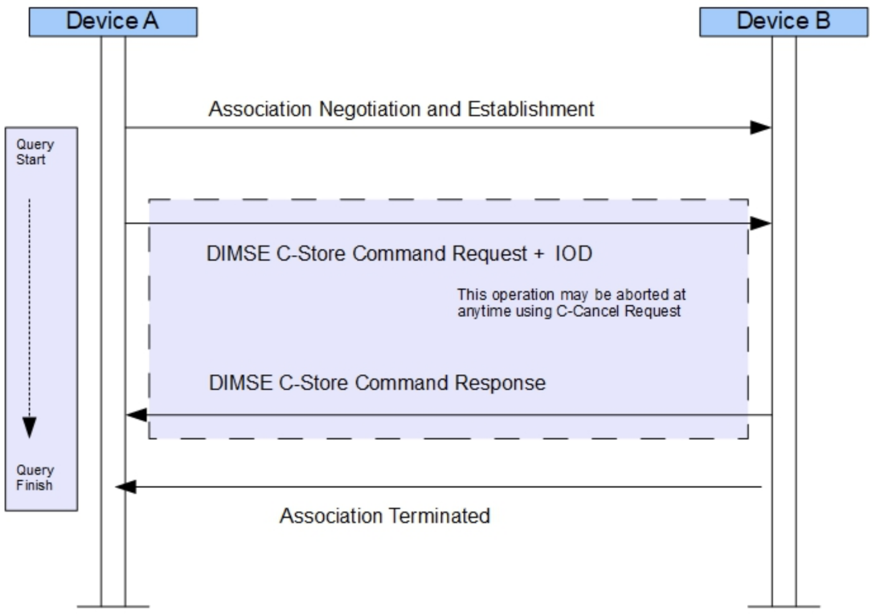

Device A
Device B
Association Negotiation and Establishment
Query Start
DIMSE C-Store Command Request + IOD
This operation may be aborted at anytime using C-Cancel Request
DIMSE C-Store Command Response
Query Finish
Association Terminated

Figure 2.5: Sequence diagram of DIMSE C-Store Operation [9]

- **Query-Retrieve:** It is used for querying the DICOM archive about its contents and eventually retrieving a specified portion of said archive to a DICOM SCU node. This service has two different phases:
  - **Query phase:** in this phase, the querying DICOM node (acting as a SCU) enquires another DICOM node (working as a SCP) about its contents and waits for a query response. This inquiry can be restricted by using search parameters.
  - **Retrieve phase:** if the query response informs of a success, the querying node may request that some archive objects be retrieved. This operation implies the transfer of DICOM data present in DICOM Query/Retrieve SCP to the DICOM Query/Retrieve SCU; this is accomplished through the above-referenced service, the DICOM Storage service.

  This service is usually implemented at a protocol level using C-FIND and C-MOVE messages; some methods use C-GET instead of C-MOVE. In the query phase, the SCU sends a C-FIND-Request (RQ) message to the SCP, which may include search parameters, and waits for C-FIND-Response (RSP) messages to be sent by the SCP, with each response conveying items that match the query search parameters. In the retrieve stage, the querying DICOM node sends a C-MOVE-RQ message to the SCP, which informs the SCP of the items to be transferred. This request message will trigger the transfer using the DICOM storage service. During the transfer, the SCP will send one or more C-MOVE-RSP messages updating the SCU of current status of the retrieve operation.

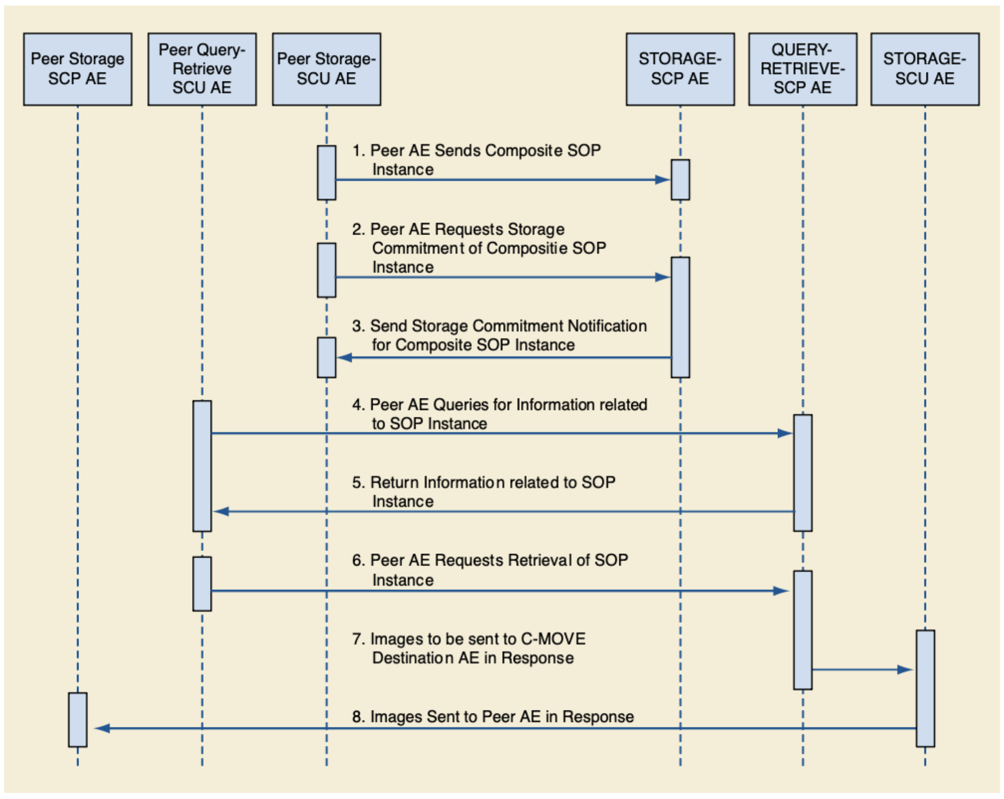

Peer Storage SCP AE
Peer Query-Retrieve SCU AE
Peer Storage-SCU AE
STORAGE-SCP AE
QUERY-RETRIEVE-SCP AE
STORAGE-SCU AE
1. Peer AE Sends Composite SOP Instance
2. Peer AE Requests Storage Commitment of Compositie SOP Instance
3. Send Storage Commitment Notification for Composite SOP Instance
4. Peer AE Queries for Information related to SOP Instance
5. Return Information related to SOP Instance
6. Peer AE Requests Retrieval of SOP Instance
7. Images to be sent to C-MOVE Destination AE in Response
8. Images Sent to Peer AE in Response

Figure 2.6: Sequence diagram of Query/Retrieve Service Operations [10]

2.2.3 DICOMweb

DICOM Standard for web-based medical imaging (DICOMweb) is not a new communication protocol but a "web view" in DICOM. This standard specifies a recent web-based service for accessing and presenting DICOM persistent objects and the mechanism used for viewing and processing these objects representing them via Hyper Text Markup Language (HTML) pages or Extensible Markup Language (XML) documents through HTTP/HTTPS (Hyper Text Transfer Protocol/Hyper Text Transfer Protocol Secure) protocol, using DICOM UIDs. The first proposed service was Web Access to DICOM Persistent (WADO) that enables a Web Client System to retrieve DICOM Persistent Objects managed by a Web Enabled DICOM Server [11]. This extension of the DICOM standard offered new possibilities for web-based PACS solutions. However, WADO does not provide storage or search features since it can only download a DICOM object if its unique identifier is known.
Recently, the DICOM Standard proposed a set RESTful Web-based services for common medical imaging operations:

- **WADO-RESTful Services (RS):** RESTful version of WADO.
- **Store Over the Web (STOW)-RS:** as the name implies, allows for storage of DICOM objects over the web to a Web Enabled DICOM Server in a RESTful manner. UIDs must be provided.
- **Query-based on ID for DICOM Objects (QIDO)-RS:** the name is also self-explanatory allows searching in a server based on the hierarchical levels of DICOM (using PatientID, or SeriesUID for example). It returns the list of queried objects that correspond to the search parameters.

2.3 Relational Databases

A relational database allows storing and retrieving data points based on pre-defined relationships. Information about related objects is stored in tabular structures. Tables can be divided into two dimensions with columns defining the different attributes of the objects and their respective data types, and rows populated with the different objects that share this characteristic.
Relations between tables can be expressed using keys (primary and foreign) to retrieve information about related data between tables, the core principle that establishes the term: *relational database.* The software systems that handle the storage, retrieval, and maintenance of data in a relational database are known as RDBMS, the great majority of these systems use Structured Query Language (SQL) as the domain-specific language responsible for database queries and data manipulation and maintenance. The four basic operations of querying, maintaining, and manipulating the data are called CRUD (Create, Retrieve, Update and Delete).
The data model of RDBMS, where data is typically structured with a fixed schema (defined by the table layout and with columns sharing a fixed data type) has some advantages and disadvantages to non-relational databases. The most relevant advantages are:

- **Offers complex queries using SQL language:** The SQL language offers programmers the ability to retrieve data and express relations from multiple tables in a single query, while implementing the most fundamental operations, i.e. the Create, Retrieve, Update and Delete (CRUD).
- **Reduced Data Redundancy:** The relational model most often involves the normalization of the data, which reduces data replication and the amount of redundant data.
- **Security:** Since RDBMS have been the industry standard for some time, security mechanics for this type of database have the benefit of time on their side being more developed and tested than their newer counterparts. RDBMS can also restrict data access by having different types of users with different access to a various tables.
- **Data model expansion:** If there is a need to express a different entity and its relation, expanding the data model is simple requiring only the creation of a new table and adding the necessary relation to the existing entities.

Some of the limitations imposed by choosing RDBMS are:

- **High costs associated with changing the data model:** Despite the ease of execution, altering the data model tends to be relatively expensive since it requires data migration to accommodate the changes.
- **Scaling:** Improving the amount of data that can be stored and the speed of read, write operations in a relational database is usually done by running it in a more powerful machine (vertical scaling). However, this is only sometimes possible due to the ever-increasing cost of upgrading the machine and technological limitations.

  Eventually, the only way to scale is horizontally, such as by distributing the data and the computational effort across multiple machines. However, RDBMS were not designed with data partitioning and distribution in mind, and some operations do not perform well (e.g. join), so this process is complex and requires a lot of knowledge and manpower.
- **Not suitable for unstructured data:** The need to normalize and structure the data into tables makes the process of working with unstructured data difficult, requiring knowledge of complex adaptation techniques.

2.4 Non-Relational Databases

Non-Relational databases, commonly referred to as NoSQL, as the name implies describe databases that store data in a non-tabular manner. They became popular with the emergence of Web 2.0 and the Big Data era, where distinct models appeared to satisfy new use case requirements. The most popular NoSQL by the model are MongoDB (Document-oriented), Redis(Key-Value), Elasticsearch (Search engine), Cassandra (Column-based), and Neo4j (Graph Database)[1]. Most of them share the characteristic of supporting a non-fixed schema

[1]https://db-engines.com/en/ranking

allowing data storage from unstructured to structured data. This results in models with lower data consistency, however through high-level features and protocols these solutions provide data reliability and availability [12]. These types of databases were designed for handling large volumes of data and low latency requirements, this results in the workloads of these new systems being dominated by high-throughput append-style inserts and read accesses in support of point or range queries [12]. NoSQL groups multiple ways of structuring data from:

- **Column-oriented:** In contrast with RDBMS where the data is stored row-oriented (record stored one after another), this type of database stores each database table column separately, with attribute values belonging to the same column stored contiguously, compressed, and densely packed [13]. Data stored in the same column is often accessed together, if the system is designed in a way to avoid scattered reads or updates, performance can be greatly improved in comparison with RDBMS.
- **Document-oriented:** Related data is stored in document format, within this type of database data tends to be denormalized, semi-structured, and stored hierarchically (like the structure of a XML file). Relations between different collections of documents are represented by a collection referencing to another collection [14].
- **Graph:** Data is structured in a way that instance of data can be modeled as a graph where objects are represented by nodes, that might have associated attributes, and the relationship between entities is represented by edges between the nodes, that also can have attributes associated. Graph-oriented operations and type constructors can express data manipulation and integrity constraints can be defined over the graph structure [15].
- **Key-Value:** Data is stored as a collection of key-value pairs with the key serving as an unique identifier. They tend not to have well-defined data types, which lends itself well to unstructured data. This simplicity in structure when configured correctly allows for a high throughput, with constant times for finding and retrieving data.

NoSQL databases due to their versatility in data modeling have some advantages over RDBMS, which are [16]:

- **Wide selection of data models:** The data model can be selected from the different types of NoSQL databases to fit the requirements imposed by the data type that is being stored.
- **Highly scalable:** NoSQL solutions were designed with scalability in mind, by supporting data partitioning allowing for horizontal scalability and ease of distributing data across different machines (usually called nodes).
- **Reduced necessity for a database administrator:** Due to the higher simplicity of the data model compared to RDBMS, the need for complex design or implementation decisions is reduced. NoSQL solutions also tend to implement features such as automatic repair and data distribution natively, reducing the necessity for an admin. Some solutions such as Cassandra and Riak have built-in tools to handle hardware failure.
- **Better performance in Big Data Scenarios:** Since the modern context of NoSQL solutions was developed to tackle Big Data problems, the speed and efficiency are superior to that of traditional databases.

- **Mostly open-source**

However, NoSQL databases also have disadvantages when compared with traditional database solutions, such as [16]:

- **Immature**: Compared with traditional database systems, NoSQL solutions are in their infancy, resulting in less support. Since these solutions are still evolving some already implemented features in RDBMS are still in development, so they are comparatively less stable, have fewer compatible tools, lack a standard interface, and are harder to maintain.
- **No standardized query language:** The lack of a standardized query language requires higher effort from programmers to query the database. Complex database queries are harder to execute than in a traditional database.
- **Atomicity, Consistency, Isolation, and Durability (ACID) transactions are usually not supported:** This results from the design decision of having data models focused on scalability and performance over data consistency, some solutions offer tools for assuring data consistency. However, some require active attention from programmers to achieve it. NoSQL solutions are based on the Basically Available, Soft state and Eventual consistency (BASE) model [17].

2.5 PACS

PACS came to exist due to the necessity of responding to the specific challenges posed by the healthcare industry, since we are dealing with medical information these systems must assure data security, permanence, and availability. The volume of data generated in medical imaging is substantial and needs to be considered. To meet the requirements imposed by the institutions, PACS need to be performant, scalable, and fault-tolerant. Maintaining a consistent quality of service requires institutions to invest in the maintenance and upgrades of their hardware infrastructure due to the ever-increasing volume of data. Anderson and Flynn define PACS as “high-speed, graphical, computer network systems for the storage, retrieval and display of radiological images” [18]. PACS acquires medical images digitally from several modalities, the outputs of different imaging machines in the radiology department. These images are stored in central data repositories or databases and are made available upon request by, for instance, referring clinicians. In modern PACS, medical images can be made available for viewing throughout the entire hospital, remotely for off-site physicians or even shared with other institutions using secure broadband connections [19]. The adoption of PACS replaces the need for manually storing, organizing, and retrieving film jackets, this should increase the efficiency of the healthcare professionals’ workflow.

The most utilized information standard to store and transfer medical images is DICOM. Medical imaging workflow can be divided into three major stages, imaging acquisition from devices (modalities), storage of said information in a digital archive, and the access of this information by a medical professional at a viewing workstation [21]. The acquisition of the images is made by specific machines commonly referred to as modalities, which are responsible for detecting the physical quantity that is being measured or observed, the

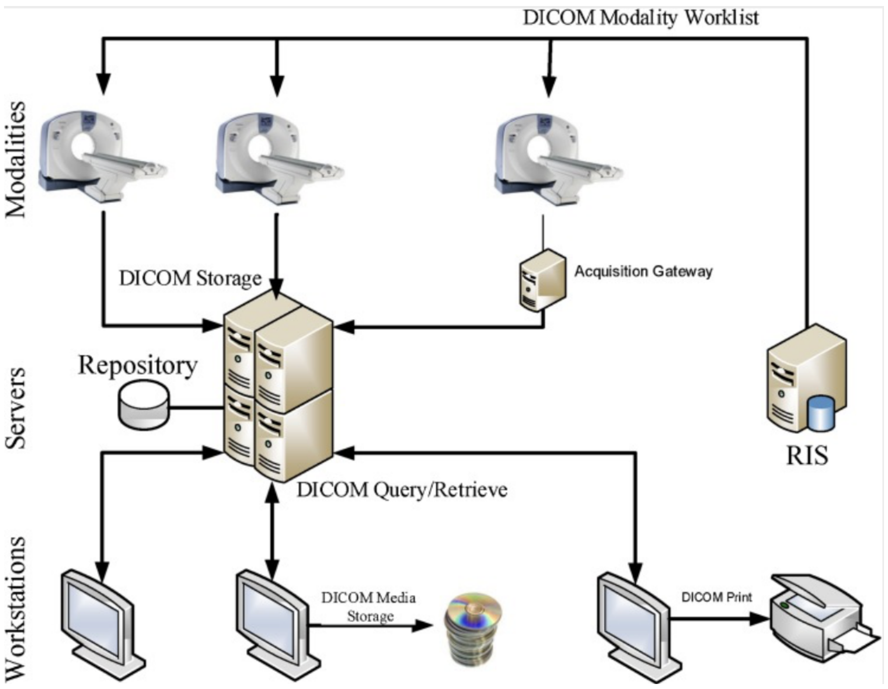


Figure 2.7: PACS Architecture Example [20]

conversion of the measurement into a digital signal, the preconditioning of said signal, and its digitization. Distribution as the name implies is the process of moving the images and their associated data from a PACS node into the destination where the information is required, this is achieved based on the process defined by the DICOM standard. The advantage of introducing PACS to deal with distribution is the ease of sharing said medical information across institutions, allowing for distributed workflows in clinical and academic research, while minimizing the possibility of misfiling or losing film jackets. To maintain an acceptable quality of service communication delays must be kept to a minimum. Visualization also as the name implies is the part of the process where the end-user utilizes a viewing station, also known as a workstation, which is equipped with specifically designed display software to allow for multi-modalities image communication, examination, and image navigation while also permitting image processing and manipulation and easy access to relevant metadata. Since these workstations are used for diagnostic purposes the quality of service must be high to avoid potential life-threatening mistakes, so most of these stations have multi-portrait mode high-resolution monitors coupled with powerful computers with large memory and a fast CPU [21]. As mentioned before dealing with medical information imposes some limitations to PACS implementations, ranging from a data security standpoint to performance marks that need to be met for quality of service assurance. However, some of these limitations are conflicting, so when designing a PACS architecture, trade-offs must be made between performance and provided features. The three prevailing PACS operation concepts can be exemplified in current clinical PACS architectures [22]:

- **Distributed:** In this model, after image acquisition, the images are sent to previously

designated workstations, and after being read they are sent to the image archive, in some models there can be an intermediary step where images are sent to the archive and only then distributed. The workstation can query and retrieve images from the archive, with images being locally stored until they are deleted by the user or specified system rules. Due to this local storage if the connection to the archive is compromised, the workflow does not need to be interrupted, and since the information tends to be available in multiple machines the likelihood of losing information is diminished greatly, while being less dependent on the network for speed. However, because multiple machines have access to the same study, problems coordinating access to exams can arise, resulting in less efficient workflows.
- **Client-Server:** In this model, after image acquisition, the data is sent to a central image archive, which functions as a server with the workstations being its clients. When a workstation requests an image the server makes that image available to that workstation, locking access to that image while it is being viewed, contrary to the distributed model, the image is only present in the client when it is being accessed, and any changes to the data are saved to the archive server. The quality of service of this model type depends on network capacity, with the server being a single point of failure.
- **Web-based:** This model has become prominent due to allowing acess to the PACS on-site or at home for greater flexibility in a physician workflow since it enables either. The web-based model is like a client-server model, with the server being a central archive integrated into a data center made available to its clients through a Web interface. The performance of this system is normally gatekept by the quality of the network connection. Still, the quality of diagnosis can be impacted since this type of model allows for lower-end hardware to be used which might not meet display quality requirements and off-site devices might not have locally installed software that bolsters system functionality.

With the ever-increasing hardware capabilities, medical imaging equipment has become more powerful and less expensive, allowing a greater number of medical institutions to generate their own medical images. However, implementing a traditional PACS is still a resource-intensive prospect. Considering the advances in web-based solutions and cloud-base computing coupled with the high cost of a PACS underlying infrastructure, the current context of the industry shifted to outsourcing this infrastructure to an off-site data center. Due to the vendor-neutrality of PACS repositories, upgrading its capability has become easier due to this outsourcing (only requiring renting more machines). Out-sourcing combined with a web-based PACS solution, also allows for easier data sharing between different institutions since the data center can service multiple sites.

PACS can also be divided by the purpose they serve into two types: institutional or traditional PACS and research PACS. The latter tends to focus on offering big flexibility in the types of searches provided to better fit the requirements imposed by clinical research, allowing for repository-wide searches. The former focuses on reducing the operational time window with higher performance at accessing the required data at the cost fewer content discovery features. The institutional PACS also has high-security requirements that must be met to assure patient confidentiality.

2.5.1 Storing pixel data in a PACS

Traditional PACS solutions usually store only the associated metadata in the database system while storing the medical images, and pixel data, in a file system or a cloud-based storage system, there are associated advantages and disadvantages. This approach tends to allow for more extensive queries since more modality-specific metadata is directly stored in the database system, this can be very useful for example, to monitor patient x-ray dosimetry, radiographic procedures, and image quality. However, looking forward to the paradigm change due to the advantages that a more distributed PACS provides, pixel data not being stored in the database system comes with problems of data replication and synchronization that tend to be complex and work-intensive problems to solve in this traditional paradigm, this is also exacerbated by the volume of data produced by medical institutions that can easily reach the tera or even petabytes of data.
Storing the pixel data in the database system, especially if the database solution chosen was developed with distribution in mind, will mitigate of this data replication and synchronization issues. However, the pixel data cannot be stored whole, since some of these files can reach gigabyte size, so the pixel data will need to be chunked which means that to retrieve and reassemble the image within acceptable time frames, the read access of the data base solution chosen will have to scale linearly.

2.6 Related Work

Analyzing the current context of medical imaging information systems, the need for a system that is easily scalable, distributed, and provides fault tolerance measures seems evident. There has been work to develop and evaluate systems that meet these requirements. Some of these studies compared a hybrid relational and XML model supported through a native XML database engine integrated with a relational database engine, with the XML model being responsible for storing the metadata that cannot be modeled in a relational model, with a MongoDB-based solution. They concluded that when querying on the hierarchical levels of the DICOM structure (Patient/Studies/Series/Image) the RDBMS showed better performance. In contrast, the MongoDB solution outpaces the RDBMS in queries that involve metadata. The MongoDB solution outperformed the hybrid relational solution in image loading throughput across all configurations [23].
Some papers also compared different NoSQL solutions against each other, MongoDB across multiple studies seems to be preferred over other solutions (especially when the focus of the study is to measure performance when reading metadata usually not easily accessible in traditional PACS solutions). However, all solutions have the edge when compared with traditional RDBMS when it comes to scalability and distribution (document-based databases also seem to have potential uses in data mining applications such as dose monitoring, quality assurance, and protocol optimization) [24][25][26]. Other papers focused on the analysis of

column-based databases, specifically Cassandra Apache, where compared with MongoDB the latter showed slightly better performance when it came to storing medical images [27], and another study concluded that Cassandra is not suitable for saving metadata information [28]. Dicoogle is a project that has been developed by UA.PT Bioinformatics in partnership with BMD Software,it is an Peer to Peer (P2P) open-source PACS archive, that is highly modular. The base version functions as a file system-based PACS where all metadata of the DICOM files can be indexed using Apache Lucene (a free text search engine), allowing extend content discovery services. The base solution is highly efficient for retrieving metadata information outperforming traditional PACS solutions. Due to the ease of adding features to the platform, some NoSQL solutions were implemented using the Dicoogle platform, with the MongoDB solution being the one that showed the most promise [29][30][26].

CHAPTER 3

Architecture Proposal

3.1 Requirements

Medical imaging storage comes with specific requirements, not shared by general database systems. This section will focus on the main requirements, considered when developing the proposed solution. The system's requirements can be divided into functional and non-functional requirements. Functional requirements define what the system does and what functions and features it provides. Non-functional requirements describe the general properties of a system, for example, if the system needs to be scalable, the availability of the system, and the amount of elapsed time for an acceptable task.

3.1.1 Functional requirements

The functional requirements that were identified were the necessity to store the obligatory DICOM fields, all the metadata present in the DICOM file and the pixel data in the chosen database system (most PACS rely on a underlying file system to store the pixel data due to their extensive size). The storage operations should maintain data integrity, for example if an error occurs while inserting a study into the database all the information spread between different tables related to that specific study should be corrected. The database system should also allow for querying along the required fields for a well-formed DICOM file, return the DICOM files that match the filters of the query and display the required fields of the DICOM files. The developed solution should allow for the an SCP-SCU pair, that establish communication DICOM machines involved, and where all the relevant information exchanged in this communication is effectively stored into the selected database.

3.1.2 Non-Functional Requirements

The most important non-functional requirement is the elapsed time the system takes to complete a task, which is most significant in querying the database (long response times to a query request will negatively impact physicians' workflow).
Other requirements are that the system must be designed to allow for horizontal scaling (as

referred before, horizontal scaling is the ability to scale by adding more machines instead of more capability to the existing machines), support geographically distributed operations, and have a fault-tolerant nature. Optimizing the disk space usage will also be important for a effective system with efficient resource usage in mind.
Since we are working with medical information, privacy is of the utmost importance with security features needed to be implemented to protect sensitive data.

3.2 Architecture

This section describes the general aspects, most significant elements, and other general proprieties used to design, model, and implement the proposed solution. The system architecture can be divided into two components: the ones required to establish the communication between two DICOM machines involved in the storage process (i.e. SCP and SCU roles); and the Cassandra Cluster that will function as the chosen database system. The SCP functions as C-Store server that responds to the storage requests sent by the SCU functioning as a client. The SCP communicates with the Cassandra Cluster for storing and retrieving data from the system database (i.e. Cassandra-based). Cassandra and the SCP communicate using the Cassandra binary protocol over TCP. The Cassandra Cluster comprises of three instances, each in a different machine. The machines run on Ubuntu 22.04.2 Latest Stable Version (LTS) operating system and have 4 Gigabytes (GB) of Random Access Memory (RAM) with 60 GB of total memory, the Python version is 3.10.6 and the Apache Cassandra version is 4.0.6.

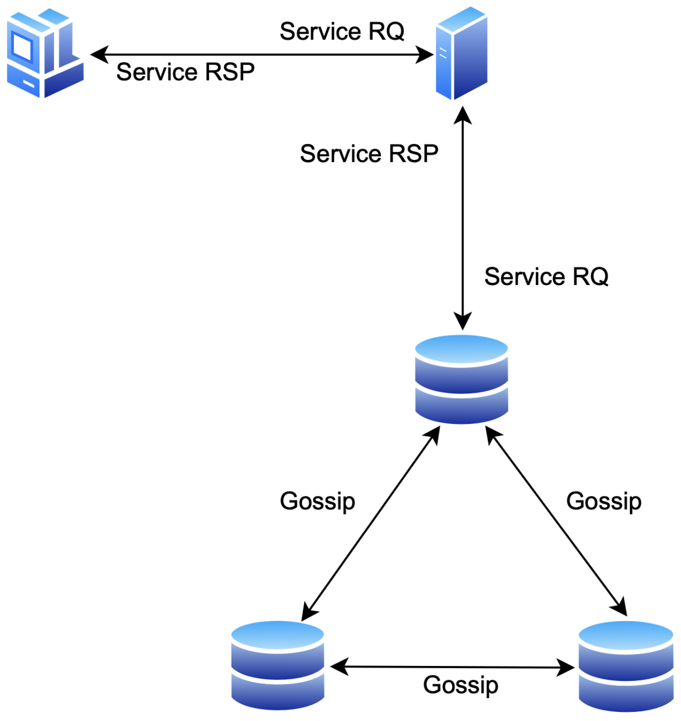


Figure 3.1: System Architecture

3.3 Database Approach

With the solution's requirements in mind, two NoSQL technologies came to the forefront Apache Cassandra and MongoDB. MongoDB presents some advantages over Cassandra mainly the flexibility due to being a document-oriented database, offering a richer data model when compared with column-oriented databases (i.e. Apache Cassandra), a more efficient secondary indexation tool, higher read speeds when Cassandra does not use the primary key to respond to the query and MongoDB offers a higher data consistency at the cost of data availability. Apache Cassandra comes with their own advantages relative to MongoDB, Cassandra does not possess a single point of failure due to their peer-to-peer architecture while MongoDB has a single point of failure due to its master-slave architecture, Cassandra also has a higher write throughput relative to MongoDB, it also offers better data availability at the cost of data consistency (which can be addressed by manipulating Cassandra configuration to assure better data consistency). Due to Apache Cassandra being developed with geographically distributed data and highly scalable application in mind, it offers a close to an out-of-the-box solution to this type of environments, with the programmer not needing to worry about the complexities of implementing these features (MongoDB also supports sharding, however, it requires some user attention to implement it, while in Cassandra the sharding is automatized).
Apache Cassandra was the selected database technology since a distributed medical imaging scenario requires the system to be the most fault-tolerant possible (due to its peer-to-peer architecture Cassandra does not have a single point of failure), a high write throughput (to handle multiple institutions producing medical information that tends to be of a significant size), high availability of the medical data is also of the utmost importance and the abstraction of the complexity of a system that supports horizontal scaling. In Cassandra, it is also possible to create indexes for the mandatory DICOM fields for reading performance in the production environment.

3.4 System Configuration

The configuration of the SCP/SCU pair and the Cassandra Cluster is implemented using a settings file. This settings file defines which IP address and port the SCP/SCU pair should communicate on. It also defines the database keyspace name, cluster name, IP address of the seeds nodes, the type of replication strategy, the replication factor and consistency factor for the Cassandra Cluster, as well as the listen address and Remote Procedure Call (RPC) address for each Cassandra node. By default, the replication strategy is *SimpleStrategy* with a replication factor 3. The listening address and RPC address should match the IP address of the machine the Cassandra node is being run on, and should match one of the IP addresses of the seed nodes. The SCP/SCU pair should also be running in one of the machines where a Cassandra node is running.

3.5 Cassandra-based PACS

3.5.1 DICOM Format and Cassandra Schema

Since Apache Cassandra is a column-oriented database, its data model is less flexible than other types of NoSQL databases. DICOM file metadata varies with the modality of the medical image, so it is only possible to store mandatory metadata using a fixed database schema. However, Cassandra allow storing structured schema-free documents as text or blob attributes. To the end of storing all data present in the DICOM file, three tables were created inside Cassandra: one table containing all required fields, the most commonly queried fields, and the total number of frames of the image, with the primary key being composed of fields that uniquely identify the different hierarchical levels of a DICOM file (Patient, Study, Series, Image); a second table that stores all non-pixel data information contained in the DICOM file that is then serialized into a JavaScript Object Notation (JSON) file, with this JSON being associated with a SOP Instance UID (a field that uniquely identifies a DICOM file at the image level); and a third table where the pixel data is stored, where the pixel data present in the file is divided into equal sized chunks. This table is composed of the SOP Instance UID, the frame of the image, the chunk number, and the pixel data information, with the partitioning key being composed of the first three fields (the chunk number is part of the primary key to avoid hot-spots when querying for pixel data).

3.5.2 Cassandra Tables and Prepared Statements

After configuring the Cassandra Cluster and the keyspace, the next step to implement the database solution is creating the tables that will be used. As referenced above three tables are created, the Cassandra python driver was used to send the Cassandra Query Language (CQL) query to the Cassandra Cluster, the figure 3.2 describes the different tables.

Using Cassandra, we can take advantage of the prepared statements feature to handle common database usage patterns such as inserts and some queries. For instance, when retrieving pixel data, multiple retrieves are often necessary due to the chunking of the pixel data to get the complete image. Therefore, prepared statements are utilized to improve efficiency in such cases. Prepared statements execute a pre-parsing of query that allows the parsing to be skipped, improving read performance.

3.5.3 Cassandra Secondary Indexes

Since Cassandra by default only optimizes queries where the search filters match table columns that are present in the primary key. To improve performance on queries where the primary key is not included, some secondary indexes were created to ameliorate the performance of this type of queries (secondary indexes where only created for the table that stores the required and most commonly queried DICOM fields). For example, it was created a secondary index for *PatientName*, to allow for a more efficient query when a physician wants to retrieve all the exams of a patient using their name instead of the *PatientID*.

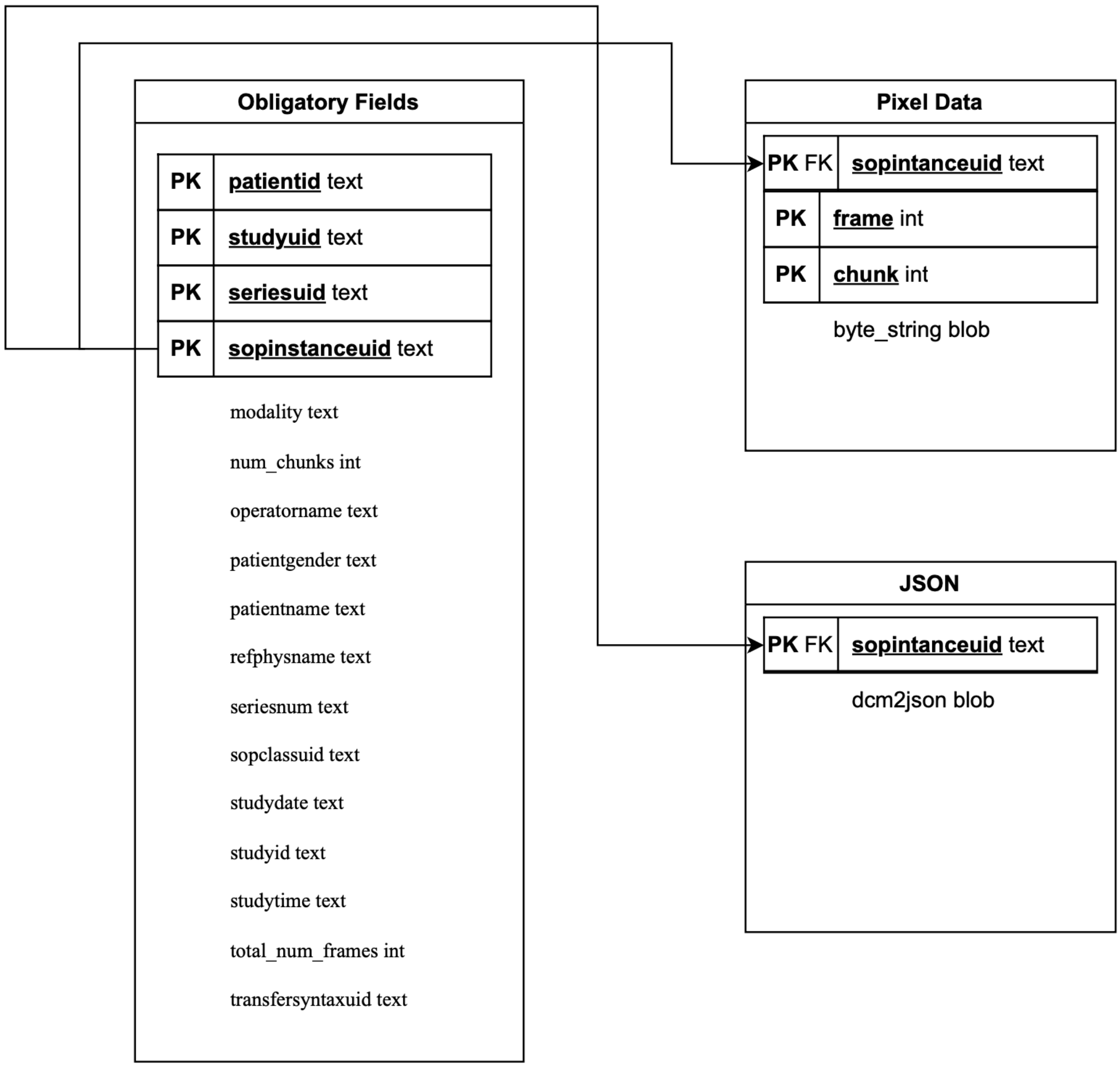


Figure 3.2: Cassandra Schema

3.5.4 Data Storage Process

When the SCU sends a C-STORE-RQ to the SCP, if the association between the machines is accepted, the SCP tries to store the DICOM file into the Apache Cassandra Cluster. The first step of this process is to extract the relevant information from said file. To this end, the *Pydicom* library is used to parse and read the information present in the DICOM file to be stored. The values of the required and most commonly queried fields along with the total number of frames of the image are temporarily stored in a python dictionary.

After extracting the values to be stored in the first table, the pixel data is the next information to be extracted, depending on the number of frames of the images two different libraries are used. If the image only has one frame, the library used continues to be *Pydicom* saving this information in a list of equal-sized chunks. However, if the images as multiple frames the library used is *Highdicom*, since *Pydicom* warns against using *Dataset.PixelData* when handling multi-frame images. In this case, the pixel data is stored in a list composed of multiple lists, with each list representing a frame. Each of these extracted frames is chunked into equal-sized chunks, this is done with the intention of avoiding bottlenecks caused by the existence of hot spots in the database; by chunking the different frames and making the chunk number part of the primary key due to the way the Cassandra distributes the data across the Cluster the creation of hot-spots can be avoided. After extracting the pixel data, the latter is deleted from the *pydicom.dataset*, which is then serialized into a JSON file (the pixel data is removed from the dataset to minimize the size of the JSON for a matter of efficiency).

3.5.5 Storage Process

After parsing the information, the next step of the process is storing said information in the database. This is done by a function that receives as arguments the Cassandra session that was started when the SCP calls the main function. The Python dictionary mentioned above contains the obligatory and most often queried metadata fields, the JSON file containing all the file metadata, and the list of lists containing various equal-sized chunks that have all the DICOM file's image information. The function feeds the arguments passed to the insert-prepared statements to insert the data into the database. Three prepared statements are each responsible for storing the above-referenced data in the three respective tables. In the case of the pixel data two cycles are required to iterate over the list of lists to properly identify the image chunks by frame and chunk number and store them correctly; the other two tables store directly the arguments passed for the function into the database.

Prepared Statement	**CQL Command**
Insert into pixel data table	INSERT INTO + keyspace + .pixeldata_table (sopinstanceuid, frame, chunk, byte_string) VALUES (?,?,?,?) IF NOT EXISTS;
Insert into json table	INSERT INTO + keyspace + .json_table (sopinstanceuid, dcm2json) VALUES (?,?);
Insert into obligatory fields table	INSERT INTO + keyspace + .test_table (modality, patientid, patientname, refphysname, seriesnum, seriesuid, studydate, studyid, studytime, studyuid, total_num_frames, sopinstanceuid, patientGender, operatorName, num_chunks, sopclassuid, transfersyntaxuid) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);

Table 3.1: Prepared Statements

3.5.6 Retrieval Process

Cassandra tables are created with a particular query in mind, and the database solution includes three primary query types. Each query is associated with a function responsible for retrieving the data from the database and processing it so that it can be return in the appropriate format. The table that stores the mandatory metadata fields, that are frequently queried, has two types of queries associated with it: one that filters the query by using only the primary key parameters and another that allows queries using secondary indexed fields.

These two query types are necessary because secondary indexes are less performant than queries that use the primary key parameters as filters. Both query types return a list of rows that match the search filters.

The table responsible for storing the JSON files containing the metadata of the DICOM file only has one associated query. It returns the *dcm2json* field (contains the serialized JSON string), which is then decoded into *UTF-8* standard and returned by the function.

The table that contains the pixel data information supports two different queries, one responsible for retrieving all the pixel data of an image and one that retrieves the data of a specified frame. To optimize the storage efficiency of the pixel data in Cassandra, the image is chunked as referenced in the section 3.5.5. Returning a whole frame, or a whole image, requires the execution of multiple retrieves to obtain an order list of chunks, that then can be joined to reform the pixel data correctly. Since the execution of the same query is done multiple times, a prepared statement was created for efficiency's sake.

When retrieving data for a single frame, the search parameters are the *SOPInstanceUID* and the frame number. The function requires an additional argument, which is the total number of chunks per frame. When the function is called, it returns an ordered list of chunks for the requested frame. The same retrieval logic applies when retrieving the full image, with an additional argument of the total number of frames required. The resulting query returns a list of lists of chunks, with each list representing a single frame and all the lists combined representing all the pixel data of the image. To obtain well-formed DICOM files, the user can use a function that combines all the queries. The function requires the search parameters and begins by issuing a retrieval request to the table containing the mandatory metadata. Rows that match the values of *sopinstanceuid*, *total_num_frames* and *num_chunks* passed as arguments are selected, the result of this query is then used to get the values to feed the search parameters of the JSON query and the pixel data query for the respective *SOPInstanceUID*. Then using the *Pydicom* library the JSON string is added to the data structure *pydicom.Dataset* along with the pixel data that was retrieved. Endianness and the explicitness of the VR are also set to their correct values. The *Pydicom* library will also be used to write a well-formed DICOM file that is then returned to the user.

All the queries are written in accordance with the CQL and can be seen in the following table.

Query Type	**CQL Command Example**
Get the obligatory fields table	SELECT * FROM test1.test_table ;
Get the obligatory fields table at a patient level	SELECT * FROM test1.test_table WHERE patientID = 'NULL_0';
Get the JSON string from the JSON table	SELECT dcm2json FROM test1.json_table where sopinstanceuid = '1.2.392.200140.2.1.1.1.4.1339200792.2072.1458821965.426'
Get the first frame and chunk of a specific image from the pixeldata table for	SELECT byte_string FROM test1.pixeldata_table WHERE sopinstanceuid = '1.2.392.200140.2.1.1.1.4.1339200792.2072.1458821965.426' AND frame = 1 AND chunk = 1;
Get the obligatory fields table at an image level	SELECT * FROM test1.test_table WHERE patientID = 'NULL_0' AND studyUID = '1.2.392.200140.2.1.1.1.2.1339200792.3476.1458821923.756' AND seriesUID = '1.2.392.200140.2.1.1.1.3.1339200792.4800.1458823129.421' AND sopinstanceuid = '1.2.392.200140.2.1.1.1.4.1339200792.4800.1458823129.422';

Table 3.2: Structure of the Retrieval CQL Commands

CHAPTER 4

Experiments and Results

4.1 Testing scenario

The testing scenario was developed to acquire metrics to verify if the requirements have been met. There are two overall scenarios: a single-node scenario, where the Cassandra Cluster only consists of one node; and multi-node scenario where the Cassandra Cluster consists of three nodes.
In the multi-node scenario, the Cassandra Clustered was configured with the replica placement strategy being a *Network Topology Strategy* with each of the different nodes being a different data center with the replication factor of 3, meaning that all of the system data is present in each node.

In both scenarios, the behavior of two types of data sets was tested. One data set is composed of 5750 DICOM files of the modality Magnetic Resonance (MR) and Computed Radiography (CR) (representing the “common” sized modalities). The other data set is comprises 460 number of DICOM files of the Slide Microscopy (SM) modality. In this modality, it is common for the number of frames to exceed ten thousand, with this data set representing large-sized modalities. For each of these scenarios, the time intervals of relevant operations were measured for both the storage and retrieval process.

4.2 Results

To analyze the performance of the developed solution from the above-described data scenario, 1000 files were selected from the data set containing the modalities MR and CR, with each of these DICOM files representing a single frame of a DICOM image, and 15 files from the data set containing DICOM files of the SM modality, composed of multiple frames samples. To evaluate single and multi-node scnearios, two different architectural setups were created. Moreover, the files containing a single frame were tested with the frames being chunked following three distinct chunk sizes, due to memory constraints the same could not be done for the DICOM files with multiple frames. Finally, temporal metrics relevant to the storage and retrieval process were measured for each implementation.

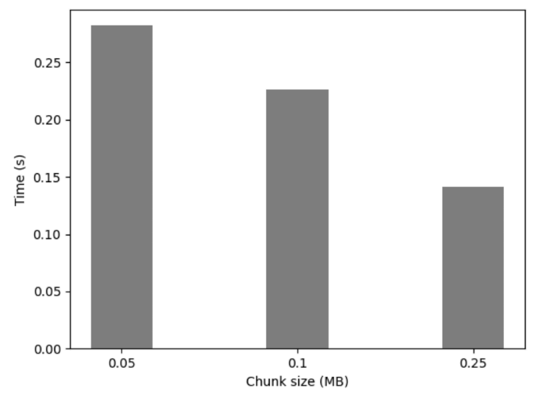
4.2.1 Fault tolerance

In the context of the system's fault tolerance, the performance of the multi-node implementation of the Cassandra Cluster was methodically tested. The objective of these tests was to verify the responsiveness of the solution when individual nodes were unavailable. The observations made revealed that the Cassandra Cluster was remarkably fault-tolerant. The testing process made the individual nodes within the multi-node Cassandra cluster unavailable. It was confirmed that the Cassandra database continued the operation as intended when one or two node were no longer communicating. It was observed that the system remained responsive and functional, provided that at least one node remained operational. The inherent fault tolerance of Cassandra underscores how robust the solution is, even when multiple nodes fail the solution ensures that the data remains available. These results affirm the system's resilience, reinforcing its suitability for mission-critical applications in medical imaging storage.

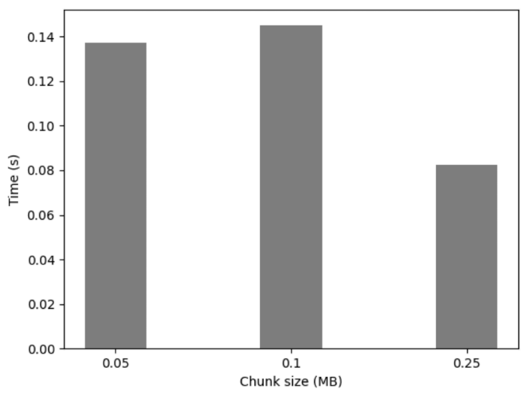
4.2.2 Storage Process

Single frame files

The graphs in the Figure 4.1 show the impact of the different chunk sizes on the storage time for the distinct Cassandra configurations, with the single-node on the left and the multi-node configuration on the right.



(a) Single-node Cassandra Cluster



(b) Multi-node Cassandra Cluster

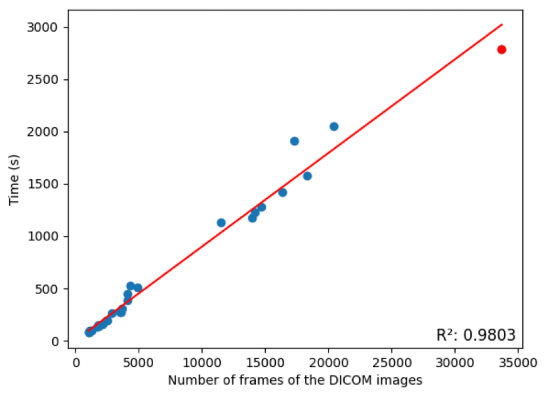
Figure 4.1: Storage time for different chunk sizes

For the single node Cluster configuration, with the larger chunk sizes, the storage process takes less time, this can be explained by how the data model was designed, the storage of a file takes more time to conclude since a lower chunk size implies a bigger number of chunks per frame, which in turns means more write operations per frame. For the 0,25 Megabytes (MB) chunk size the images stored are not divided, with a chunk comprising the whole image, this setup seems to be the optimal chunk size in the developed testing scenario. However, in a multi-node configuration there might exist a smaller chunk size where the performance start to approximate storing the whole image without chunking. Comparing the different Cassandra Cluster configurations, storing single-frame DICOM files is faster in a multi-node configuration.

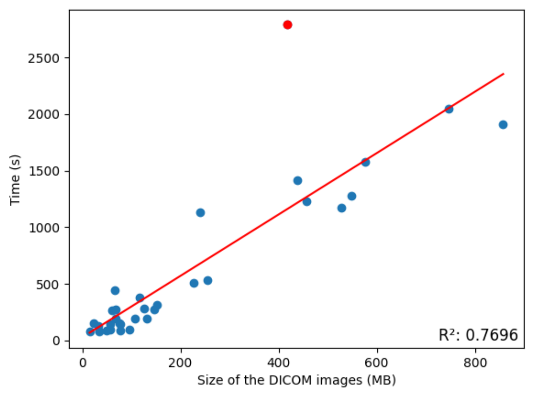
This is due to the ability of the Cassandra cluster to load-balance the write requests, meaning that it can distribute the requests across the cluster. A multi-node configuration also reduces the resource contention and should increase performance since the requests do not need to all be directed to the same node. Also, when comparing with the single-node configuration, the difference between the average time to store a file for a chunk size of 0,05 MB and 0,10 MB is much closer, this can also be explained by the ability of Cassandra to parallelize the write requests and use the processing power of three machines instead of one.

Multiple frames files

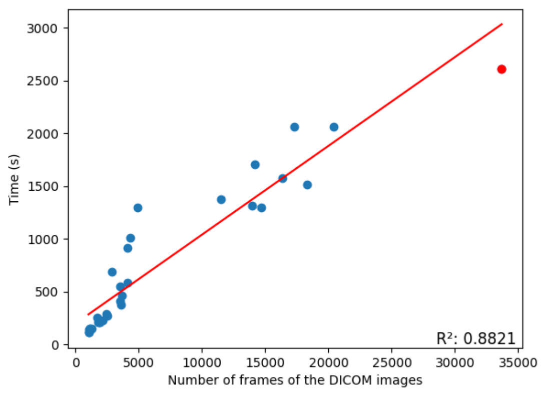
The graphs in the Figure 4.2 illustrate how the store time is impacted by the number of frames in the DICOM file versus the file size, to see what impacts the performance more. The top part of the graph refers to a Cassandra Cluster with a single-node and the bottom part shows the multi-node Cassandra Cluster:



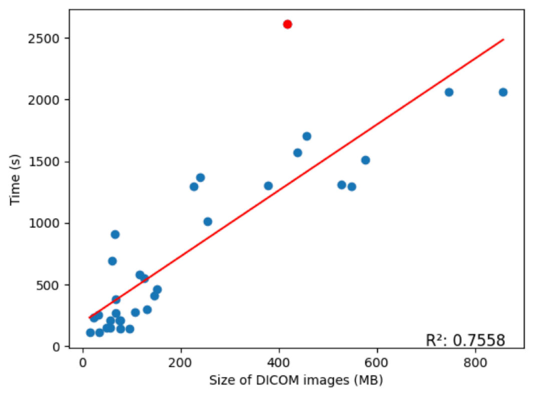
(a) Storage time across frame count in a single-node Cassandra Cluster



(b) Storage time across file size in a single-node Cassandra Cluster



(c) Storage time across frame count in a multi-node Cassandra Cluster



(d) Storage time across file size in a multi-node Cassandra Cluster

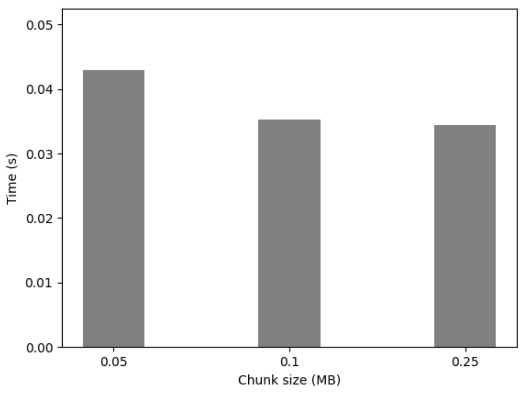
Figure 4.2: Storage time across frame count and file size

The graphs clearly demonstrate that the number of frames of the file has a higher impact on the performance of the database when compared with the size of the file, where DICOM files with a higher frame count taking longer than files of a similar size with less frames. This is best exemplified by the file highlighted by the red point, being a file that is average size wise but has the highest frame count, with this file taking the longest out of all the files. This behaviour is expected since more frames means more write operation needed to store the file. The values of the graphs in Figure 4.2a and Figure 4.2c show that the single-node implementation scales more linearly than the multi-node implementation, with the r^2 value being higher in single-node. The single-node configuration has a lower storage time for files with fewer frames. However, for bigger files, the multi-node configuration outperforms the single-node configuration. This makes sense since a higher number of frames means more operations to be paralellized and the combined processing power of the multi-node cluster to start to make a difference.

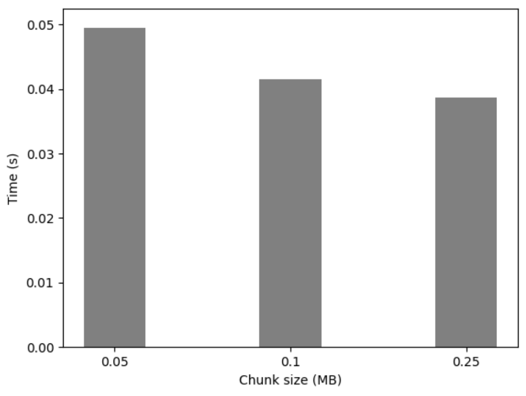
4.2.3 Retrieval Process

Single frame files

The graph in the Figure 4.3 shows the average retrieval time for a query at the image level, with the graph on the left representing the results for the single-node Cassandra Cluster configuration and the graph on the right referring to the multi-node Cassandra Cluster configuration.



(a) Single-node Cassandra Cluster

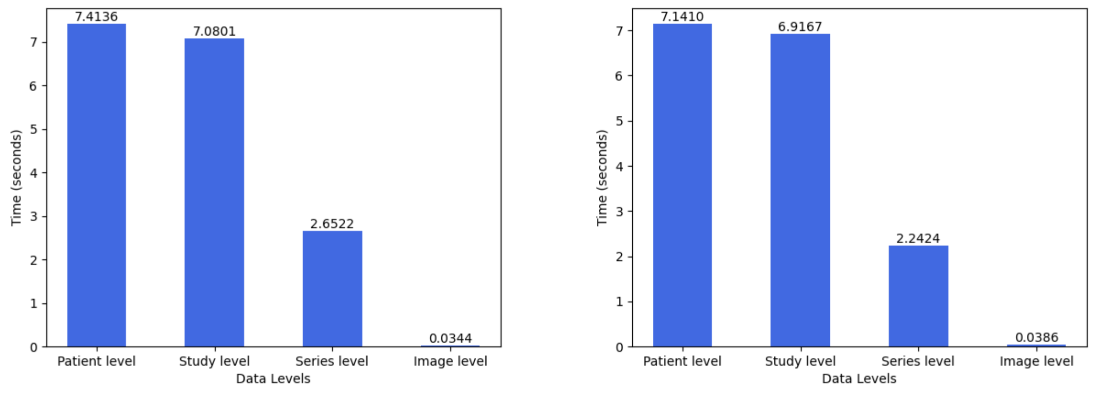


(b) Multi-node Cassandra Cluster

Figure 4.3: Retrieval time at image level for different chunk sizes

The graphs illustrate that retrieval time for different chunk sizes tends to grow with the increase in the number of chunks needed to return the DICOM file, since lower chunk sizes create more chunks per image lower chunk sizes will perform worse, with retrieving the whole image being the most performant of the chunk sizes tested. This is true for both the single node and multi-node Cassandra configuration. However, in the multi-node implementation, the smaller chunk size is less damaging to performance when compared with larger chunk sizes since Cassandra can distribute the request across the cluster. When comparing the performance of a single-node directly to a multi-node configuration, the single node performs better in this scenario since the number of operations needed to retrieve the image needs to be higher compared with the delays introduced by communications between nodes required in the multi-node configuration.

The graphs in the Figure 4.4 show the average retrieval time per data level comparing the single-node and multi-node configurations of the Cassandra Cluster with a chunk size of 0,25 MB, with the former on the left and the latter on the right. The data set is organized so that for a patient there is an average of 300 DICOM files, for the same patient different studies have an average of 150 files associated, and each series inside these studies averages 30 files.



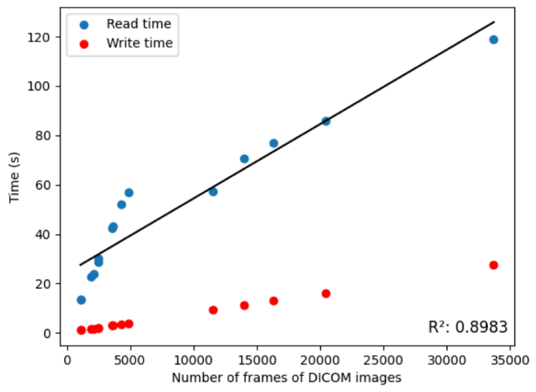
(a) Single-node Cassandra Cluster **(b)** Multi-node Cassandra Cluster

Figure 4.4: Retrieval time at the different data levels

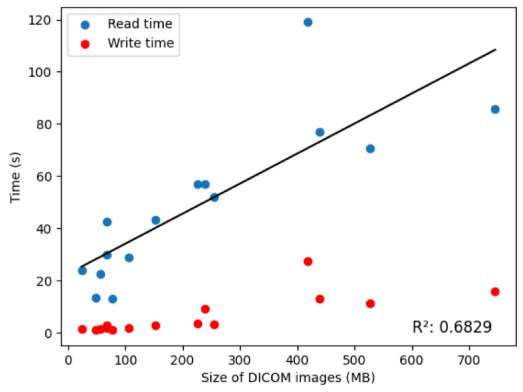
From the graphs, it can be concluded that the single-node configuration outperforms the multi-node configuration in retrieving information at the image level. However, when retrieving a larger number of files the multi-node configuration starts to outperforms the single-node configuration. The observed difference in retrieval time between the Patient and Study level was expected to be more pronounced, the small difference might be explained by the *PatientID* being the only term on the partition key, since Cassandra is designed to be highly efficient on queries along the partition key.

Multiple frames files

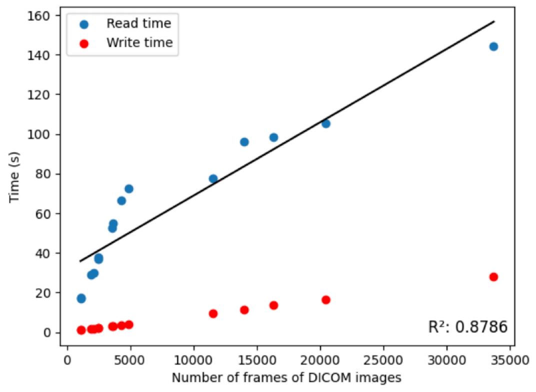
The graphs in Figure 4.5 show how the time retrieving data, represented by the blue dots, and the time spent writing this data to a DICOM file, represented by the red dots, vary with the number of frames and size of the files, with the ones referring to the number of frames on the left and the size of the files on the right. It also illustrates the impact of a single-node Cassandra Cluster configuration, in the top part of the figure, versus a multi-node Cassandra Cluster, in the bottom part of the figure.



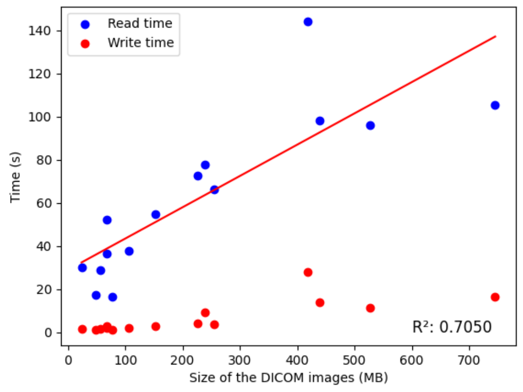
(a) Retrieval time across frame count in a single node Cassandra Cluster



(b) Retrieval time across file size in a single node Cassandra Cluster



(c) Retrieval time across frame count in a multi-node Cassandra Cluster

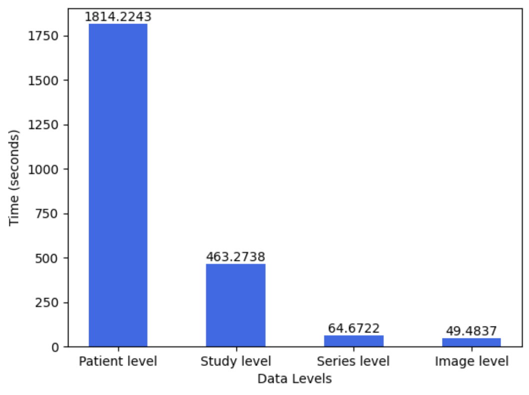


(d) Retrieval time across file size in a multi-node Cassandra Cluster

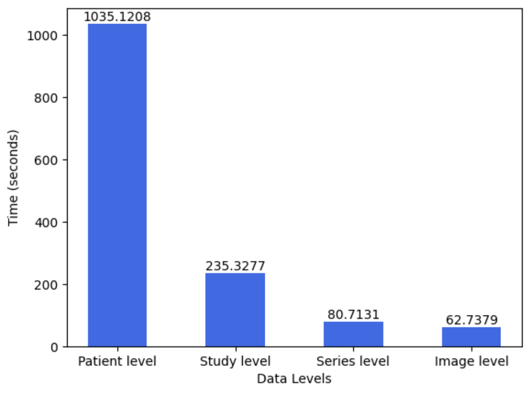
Figure 4.5: Retrieval time across frame count and file size

The graphs show that the number of frames has a higher impact on the retrieval time than the size of the files, as seen before in the storage times. When comparing the different Cassandra Cluster configurations, it can be concluded that for retrieving at the image level the single-node configuration outperforms the multi-node configuration. Again this is due to the amount of operations necessary to retrieve the image needing to be more significant to overcome the overhead associated with a multi-node configuration.

The graphs in the Figure 4.6 show the average of the queries at different data levels in both single node, on the left, and multi-node Cassandra Cluster configuration, on the right.



(a) Single-node Cassandra Cluster



(b) Multi-node Cassandra Cluster

Figure 4.6: Retrieval time at different data levels

As seen in the Figure 4.5, in single-node, the retrieval of DICOM files at the image level is faster, and the same can be said for the series level. However, as the number of operations necessary increases significant time gains are made with the multi-node configurations, almost halving the time needed to retrieve all the files at a study and patient level. As referenced before to overcome the overhead associated with a multi-node configuration a certain amount of operations is required, with number of queries being need to retrieve at series and image level not meeting the threshold to benefit from a multi-node configuration. Also analyzing the average time that the solution takes to retrieve a multi-frame image, it is clear it would negatively impact the workflow of the end-user. This maybe due to the design decision to include both the *frame* and *chunk* in the partition key of the table holding the data pixel information. If the partition key included only the *sopinstanceuid*, the retrieval of all the pixel data information of one image would probably be more efficient, coming at the cost of less data distribution.

CHAPTER 5

Conclusion

Medical imaging techniques are producing a growing volume of data that is usually stored in centralized repository. However, the increasing amount of data being generated makes it economically unsustainable to scale such repositories in the long run.

The principal objective of this thesis was to develop a database solution that would be easily distributed, fault-tolerant, scale horizontally and performant to address the Big Data problem facing the medical imaging field.

To this end a Cassandra-based solution was designed and implemented. It started by designing the database schema to store medical imaging data efficiently, then the data was treated to fit the defined schema while maintaining its integrity, and still be able of return a valid DICOM file from the information in the database. The Cassandra cluster was then configured to allow for a multi-node configuration. Then a variety of tests were conducted to evaluate the solution.

The experimental process revealed that the solution does scale horizontally, this is more evident when the amount of data is larger, and the multi-node configuration start to outperform the single-node implementation. The solution should also be easily distributed with Cassandra, since Cassandra allows the nodes to be organized into datacenters and racks, meaning we can differentiate different geographic locations by organizing nodes per datacenter and improve fault tolerance by specifying inside the datacenter to which racks does the node belongs to. This is easily configured in the configuration files, and Cassandra handles the implementation logic. It was also verified that Cassandra can handle the loss of nodes while maintaining the stored data available to the end user, proving the fault-tolerance of the solution. Performance-wise, there is a demarcation in the quality between the testing scenarios of single frame and multiple frame files. While the performance of the single-frame files meets expectations, the scenario of the multiple-frame files failed to meet the requirements with the average storage and retrieval time per frame almost four times higher than the scenario of the single frame images. This is probably due to a mixture of two factors, with the first being that two distinct libraries were used with *Pydicom* being the one responsible for the single-frame files, while

Highdicom was used for the multi-frame files. This may be addressed by transforming a DICOM file into multiple single-frame DICOM files.

However, this process can vary depending on the modality of the DICOM file and how the file was structured. It can also prove to be a difficult and time-consuming process. The second factor is the partition key for the pixel data table included the number of the frame and the chunk. This decision was made with data distribution in mind, however it forced the execution of more queries instead of being able to aggregate the query at a *sopinsntaceuid* level, with the multi-frame being more impacted by this design choice. Changing the partition key to only the *sopinstanceuid* and treating the retrieve data programmatically instead of forcing the database to do more queries would probably increase performance in this instance. Also, some results about the storage process experienced distortion. This occurred since the order of inserting single-frame and multi-frame files was significant. Storing multi-frame files first resulted in slower storage times for single-frame files due to memory constraints in the virtual machines used, this explains the results displayed in the graph of Figure 4.1b. Furthermore, the testing of different chunk sizes for multi-frame files was also affected by the memory constraints, since it would have significantly restricted the amount of files that could have been stored, potentially leading to less statistically significant test results. To enhance the robustness and the potential market readiness of the solution, adding a MongoDB module for handling diverse metadata is recommended. Since Cassandra is not the most suitable choice for handling such varied metadata, this information was stored in a JSON byte string. The MongoDB module would also add the ability to query non-mandatory metadata, further enhancing the solution's usability. Moreover, to reach more optimized configuration settings additional tests are queried, with a special focus on the replication factor and strategy. The overall solution shows promise having met most of the goals set at the thesis' inception. It should be highlighted that contrary to contemporary solutions, it does not rely on a file system to retrieve the corresponding DICOM file. Instead, all pertinent data to obtain the DICOM file is stored in the database.

References

[1] S. Sagiroglu and D. Sinanc, *Big data: A review.* 2013, pp. 42–47. DOI: `10.1109/CTS.2013.6567202`.

[2] C.-C. Teng, J. Mitchell, C. Walker, *et al.*, «A medical image archive solution in the cloud», in *2010 IEEE International Conference on Software Engineering and Service Sciences*, 2010, pp. 431–434. DOI: `10.1109/ICSESS.2010.5552343`.

[3] A. Meyer-Baese and V. Schmid, *Chapter 1 - Introduction*This chapter contains material reprinted from chapter 1 of Biomedical Signal Analysis: Contemporary Methods and Applications, by Fabian Theis and Anke Meyer-Base, published by The MIT Press. Reprinted with permission from MIT Press.**, Second Edition, A. Meyer-Baese and V. Schmid, Eds. Oxford: Academic Press, 2014, pp. 1–20, ISBN: 978-0-12-409545-8. DOI: `https://doi.org/10.1016/B978-0-12-409545-8.00001-7`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9780124095458000017`.

[4] «Dicom ps3.10 - media storage and file format for media interchange», DICOM Standards Committee. (), [Online]. Available: `https://dicom.nema.org/medical/Dicom/2016b/output/chtml/part10/chapter_7.html` (visited on 05/29/2023).

[5] DICOM Standards Committee. «DICOM PS3.10 - Media Storage and File Format for Media Interchange». (), [Online]. Available: `https://dicom.nema.org/medical/dicom/current/output/chtml/part04/chapter_6.html` (visited on 06/03/2023).

[6] «DICOM PS3.4 - Conformance», DICOM Standards Committee. (), [Online]. Available: `https://dicom.nema.org/medical/dicom/current/output/html/part04.html` (visited on 05/29/2023).

[7] «Overview: Basic dicom file structure», LEAD Technologies, Inc. (Jan. 2023), [Online]. Available: `https://www.leadtools.com/help/sdk/v21/dicom/api/overview-basic-dicom-file-structure.html`.

[8] «DICOM PS3.5 - Data Structures and Encoding», DICOM Standards Committee. (), [Online]. Available: `https://dicom.nema.org/medical/dicom/current/output/chtml/part05/chapter_7.html` (visited on 05/29/2023).

[9] S. Subramanian. «Dicom c-store implementation». (), [Online]. Available: `https://saravanansubramanian.com/dicomcstore/` (visited on 05/29/2023).

[10] «DICOM PS3.6 - Data Dictionary», DICOM Standards Committee. (2013), [Online]. Available: `https://dicom.nema.org/dicom/2013/output/chtml/part02/sect_F.4.html` (visited on 05/29/2023).

[11] G. V. Koutelakis and D. K. Lymperopoulos, «Pacs through web compatible with dicom standard and wado service: Advantages and implementation», in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 2006, pp. 2601–2605.

[12] J. Schindler, «Profiling and analyzing the i/o performance of nosql dbs», *SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 389–390, Jun. 2013, ISSN: 0163-5999. DOI: `10.1145/2494232.2479782`. [Online]. Available: `https://doi.org/10.1145/2494232.2479782`.

[13] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, «Column-oriented database systems», *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1664–1665, Aug. 2009, ISSN: 2150-8097. DOI: `10.14778/1687553.1687625`. [Online]. Available: `https://doi.org/10.14778/1687553.1687625`.

[14] R. Mason, «Nosql databases and data modeling techniques for a document-oriented nosql database», Jul. 2015. DOI: `10.28945/2245`.

[15] R. Angles and C. Gutierrez, «Survey of graph database models», *ACM Computing Surveys*, vol. 40, Feb. 2008. DOI: 10.1145/1322432.1322433.

[16] A. Nayak, A. Poriya, and D. Poojary, «Type of nosql databases and its comparison with relational databases», *International Journal of Applied Information Systems*, vol. 5, no. 4, pp. 16–19, 2013.

[17] D. Kunda and H. Phiri, «A comparative study of nosql and relational database», *Zambia ICT Journal*, vol. 1, no. 1, pp. 1–4, Dec. 2017. DOI: 10.33260/zictjournal.v1i1.8. [Online]. Available: https://ictjournal.icict.org.zm/index.php/zictjournal/article/view/8.

[18] D. Anderson and K. Flynn, «Picture archiving and communication systems: A systematic review of published studies of diagnostic accuracy, radiology work processes, outcomes of care, and cost», *Database of Abstracts of Reviews of Effects (DARE): Quality-assessed Reviews [Internet]*, 1997.

[19] R. van de Wetering and R. Batenburg, «A pacs maturity model: A systematic meta-analytic review on maturation and evolvability of pacs in the hospital enterprise», *International Journal of Medical Informatics*, vol. 78, no. 2, pp. 127–140, 2009, ISSN: 1386-5056. DOI: https://doi.org/10.1016/j.ijmedinf.2008.06.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1386505608001056.

[20] F. Valente, L. A. B. Silva, T. M. Godinho, and C. Costa, «Anatomy of an extensible open source pacs», *Journal of digital imaging*, vol. 29, no. 3, pp. 284–296, Jun. 2016, ISSN: 0897-1889. DOI: 10.1007/s10278-015-9834-0. [Online]. Available: https://europepmc.org/articles/PMC4879027.

[21] J. Zhang, J. Sun, and J. N. Stahl, «Pacs and web-based image distribution and display», *Computerized Medical Imaging and Graphics*, vol. 27, no. 2, pp. 197–206, 2003, Picture Archiving and Communication Systems 20 Years Later, ISSN: 0895-6111. DOI: https://doi.org/10.1016/S0895-6111(02)00074-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895611102000745.

[22] H. K. Huang, «Pacs and imaging informatics: Basic principles and applications», 2004.

[23] D. Teng, J. Kong, and F. Wang, «Scalable and flexible management of medical image big data», *Distributed and Parallel Databases*, vol. 37, pp. 1–16, Jun. 2019. DOI: 10.1007/s10619-018-7230-8.

[24] S. Nagappa, «Bigdata: A survey on rdbms and various nosql databases on storing medical images», Aug. 2017.

[25] D. R. Rebecca and I. E. Shanthi, «A nosql solution to efficient storage and retrieval of medical images», *International Journal of Scientific & Engineering Research*, vol. 7, no. 2, pp. 545–549, 2016.

[26] L. A. B. Silva, L. Beroud, C. Costa, and J. L. Oliveira, «Medical imaging archiving: A comparison between several nosql solutions», in *IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, 2014, pp. 65–68. DOI: 10.1109/BHI.2014.6864305.

[27] D. R. Rebecca and D. I. E. Shanthi, «Analysing the suitability of storing medical images in nosql databases», 2016.

[28] A. Savaris, T. Härder, and A. v. Wangenheim, «Evaluating a row-store data model for full-content dicom management», in *2014 IEEE 27th International Symposium on Computer-Based Medical Systems*, 2014, pp. 193–198. DOI: 10.1109/CBMS.2014.61.

[29] C. Costa, C. Ferreira, L. Bastião Silva, L. Ribeiro, A. Silva, and J. Oliveira, «Dicoogle - an open source peer-to-peer pacs», *Journal of digital imaging : the official journal of the Society for Computer Applications in Radiology*, vol. 24, pp. 848–56, Oct. 2010. DOI: 10.1007/s10278-010-9347-9.

[30] A. Almeida, F. Oliveira, R. Lebre, and C. Costa, «Nosql distributed database for dicom objects», in *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2020, pp. 1882–1885. DOI: 10.1109/BIBM49941.2020.9313430.