Universidade Estadual de Londrina

LUIS FERNANDO MILANO OLIVEIRA

A DISTRIBUTED WORKFLOW-BASED ARCHITECTURE FOR CONTENT-BASED IMAGE RETRIEVAL

Londrina
2019

LUIS FERNANDO MILANO OLIVEIRA

A DISTRIBUTED WORKFLOW-BASED ARCHITECTURE FOR CONTENT-BASED IMAGE RETRIEVAL

A thesis presented to the Graduate Program in Computer Science at the State University of Londrina to obtain the degree of Master of Science in Computer Science.

Advisor: Prof. Dr. Daniel dos Santos Kaster

Londrina
2019

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Oliveira, Luis Fernando Milano.
A distributed workflow-based architecture for content-based image retrieval / Luis Fernando Milano Oliveira. - Londrina, 2018.
55 f. : il.

Orientador: Daniel dos Santos Kaster.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2018.
Inclui bibliografia.

1. Banco de Dados - Tese. 2. Recuperação de Imagens por Conteúdo - Tese. 3. Arquitetura Distribuída - Tese. 4. Big Data - Tese. I. Kaster, Daniel dos Santos. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

LUIS FERNANDO MILANO OLIVEIRA

A DISTRIBUTED WORKFLOW-BASED ARCHITECTURE FOR CONTENT-BASED IMAGE RETRIEVAL

A thesis presented to the Graduate Program in Computer Science at the State University of Londrina to obtain the degree of Master of Science in Computer Science.

EXAMINATION BOARD

_____

Advisor: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina - UEL

_____

Prof. Dr. Pedro Henrique Bugatti
Universidade Tecnológica Federal do Paraná - UTFPR

_____

Prof. Dr. Sylvio Barbon Junior
Universidade Estadual de Londrina – UEL

Londrina, 27 de abril 2018.

ACKNOWLEDGEMENTS

I thank everyone that supported me in any way throughout the making of this work. In special, I am grateful for the support of my family, the supervision of Prof. Kaster, and for the friends who contributed in many ways, both in the work and in life. I am also thankful to both CAPES and CNPq - Projeto Universal, for the financial support.

MILANO-OLIVEIRA, L. F.. **A distributed workflow-based architecture for content-based image retrieval**. 2018. 54p. Master's Thesis (Master in Computer Science) – State University of Londrina, Londrina, 2018.

ABSTRACT

The increasing size of datasets imposes a challenge for data management systems, as people produce tons of information everyday. Not only data should be stored, but ways of fetching relevant information among in this flood of data are also in demand. Complex data, such as images, videos, and spatial data, bring an even harder challenge since they require specific retrieval techniques, such as similarity search. In this work, our first concern is to provide methods that allow users to embed their domain knowledge to build a representation of complex data that resembles their notion of similarity, which we call similarity space. Once a similarity space is defined, we also aim at providing ways for efficient distributed similarity search in datasets that are too big for being processed by centralized approaches. Our contributions are: (i) a distributed architecture that allows the definition of similarity spaces through scientific workflows; and (ii) an approach for querying similarity spaces in distributed environments. This search mechanism allows virtually any metric access method to be used as a plugin to the architecture, delegating to them the responsibility of indexing, pruning, and query processing tasks, which enables centralized indexes to be run in distributed environments. We evaluated our approaches implemented in a framework and were able to achieve linear speedup in similarity space definition activities while up to 24 times faster similarity searches when compared to a baseline sequential method.

Keywords: Content-based Image Retrieval, Multimodal Similarity Search, Data Partitioning, Distributed Computation

MILANO-OLIVEIRA, L. F.. **Uma arquitetura distribuída baseada em workflows para recuperação de imagens por conteúdo**. 2018. 54f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2018.

RESUMO

O aumento de conjunto de dados é um dos desafios que devem ser superados por sistemas de gerenciamento de dados, uma vez que usuários geram muita informação diariamente. Contudo, apenas armazenar esses dados não é suficiente. Também é necessário que sejam fornecidas maneiras de se recuperar informação relevante dentro desse oceano de dados. Dados complexos – como imagens, vídeos, e dados espaciais – ainda adicionam outro desafio, na medida em que eles exigem técnicas de recuperação específicas, como buscas por simaridade. Neste trabalho, nossa primeira preocupação é fornecer métodos que permitam a usuários incluir seu conhecimento de domínio para construir uma representação desses dados complexos que se assemelha à suas noções de similaridade, o que chamamos de espaços de similaridade. Assim que um espaço de similaridade é definido, nós também fornecemos mecanismos para a realização eficiente de buscas de dados complexos. Nossas contribuições são: uma arquitetura distribuída que permite a definição de espaços de similaridade através de *workflows* científicos; e (ii) uma abordagem de consulta a espaços de similaridade em ambientes distribuídos. Essa segunda abordagem permite que virtualmente qualquer método de acesso métrico seja usado como um plugin para a arquitetura, delegando a eles a responsabilidade por tarefas de indexação, poda e processamento de consultas, o que permite que índices centralizados sejam executados em ambientes distribuídos. As abordagens foram implementadas em um *framework* e através de uma avaliação foi possível alcançar um *speedup* linear em atividades de definição de espaços de similaridade, enquanto a performance em buscas por similaridade foi de até 24 vezes melhor quando comparada a um método sequencial.

Palavras-chave: Recuperação de Images por Conteúdo, Busca por Similaridade Multimodal, Particionamento de dados, Computação Distribuída

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

CBIR	Content-Based Image Retrieval
CT	Computerized Tomography
DICOM	Digital Imaging and Communications in Medicine
MAM	Metric Access Method
MRI	Magnetic Resonance Imaging
NHS	National Health Service
REST	Representational State Transfer
SWfMS	Scientific Workflow Management Systems

CONTENTS

1 INTRODUCTION

Technological development has brought numerous ways of both producing and consuming information, and serving many different purposes. Users produce data in their daily routine in multiple forms, by writing long essays on the web, taking pictures or making videos with their smartphones, or by using mobile applications with Global Positioning System (GPS) support to track their progress when training for a run. However, without proper ways for fetching specific pieces of data all this data produced are just stored and forgotten. With that comes the ability to remember a particular moment that represents something he or she considers special, which varies with who the user is. It can be a family picture taken ten years ago – what most of us do – or a medical record that contains a diagnosis for a rare condition – an example in a very specific context.

Searching traditional data, such as small character strings, dates, and numeric values, is well supported in Database Management Systems (DBMS). These data types have a property called *total order relation* – which states that any pair of elements in a set has a defined order, and database systems usually leverage on this property to perform efficient data storage and retrieval. However, the total order relation is not defined for most complex data domains, including images, videos, audio tracks, spatial data, large texts, and other unstructured types of data. Such data are better organized and searched through the concept of similarity. Similarity search makes it possible to query data by providing an example object, as when searching photos similar to the one a user just took, or searching for songs that are akin to a given audio fragment.

An example of such query in the image domain is: "*given the example image i_q return the three images from the dataset D with the most similar color patterns*". Figure 1 depicts an image used as the example in a query of this kind, as performed by Lire [1], a framework that provides similarity search for image datasets. In this case, the system is using a color likeness criteria for evaluating how similar two images are. A system based on similarity search techniques would be able to inspect a dataset and provide a (possibly empty) set of answers to the query. Figure 2a through 2c shows some examples of images that could be returned by the query. We will address several details about those techniques throughout this work, but this simple example already allows one to realize that, even though the images are very alike considering the mentioned criteria (a color-based one), the third image does not show a bird. The similarity evaluation should be good enough to exclude the photo with the plane (Figure 2c)from the result set and include a new one of a bird similar to the one in Figure 1. Nevertheless, to identify an ideal combination of resources providing a similarity space that resembles user perception according to a given purpose is usually very challenging.

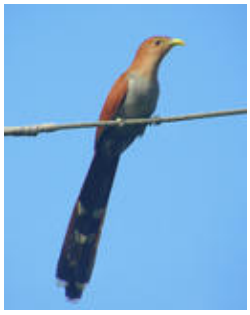


Figure 1 – Query Image. Source: Lire Project

(a) (b) (c)

Figure 2 – The set of returned answers. Source: Lire Project.

1.1 Motivation

Content-Based Image Retrieval (CBIR) is a set of technologies and techniques that target the management of image databases by using their visual content, making it possible to search by visual similarity [2]. There are many approaches for employing CBIR in domains like medicine [3], astronomy [4], security [5] and others. A possibility it brings to the medical domain, for instance, is that physicians can use knowledge from previous diagnoses in new cases based on imaging exams, supported by an information system. Despite its benefits, employing CBIR techniques for several application domains can be troublesome and complex. In order to search in a dataset by visual similarity, systems cannot use the images in their raw and unstructured form. They should be represented in a way that allows for similarity retrieval. This representation is known as similarity space in which points represent objects, and the distance between points is inversely proportional to the similarity between the corresponding objects [6]. Ideally, the similarity measure used by a system should resemble the perception of a human domain specialist. Achieving it can be a very exhaustive process, as it involves combining and tuning several complementary CBIR techniques and algorithms for image processing, feature extraction, indexing, among others, which can be overwhelming for users with no programming background.

Indeed, although to achieve suitable results using one modality (e.g., a set of color patterns in the provided example) is already complex in isolation, there is also the possibility of combining two or more modalities. For example, images may have associated tags describing their content, such as the class of object present on the scene (bird, plane,

or any other), or videos may have spatial information that is automatically attached to them at the moment they are captured. The enrichment of complex data with metadata and other complementary information and with different patterns extracted from their content greatly increases the kinds of queries users can request to systems. By using it, it is possible that a system answers very elaborated predicates, such as photos taken in Alaska, from the year 2010, and by a specific scientist that studies the bird species shown in Figure 1.

When considering systems focused on the management of image data, this level of interaction is already in the reach of regular users, especially through social media. Services like Google Photos[1], for instance, let users organize their pictures by the location where they were taken. Nevertheless, there are applications, such as scientific ones, that benefit from domain-specific approaches for managing data, mainly because their users may make important decisions by analyzing it. For instance, a radiologist employs knowledge when interpreting an image from an imaging exam that is far from obvious to the ordinary user.

Challenges in retrieval of complex data come not only from the nature of the data, but also from the amount of it already stored and from the rate of its production. The number of photos stored worldwide is expected to grow from 3.2 trillion in 2015 to 4.7 trillion in 2017. In 2017, an increase of about 9% in the number of photos taken is estimated when comparing to 2016[2]. As a more specific example, medical imaging exams of modalities such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) had average annual growth of more than 10% from 1995 to 2014 in the National Health Service in England with an ever-increasing year-by-year rate[3]. Therefore, it is desirable that retrieval systems offer support for processing and searching such volume of complex data, and distributed approaches are known to bring the benefit of both distribution of storage and parallelization of query execution [7], besides speeding up other operations involved in this kind of application.

1.2 Objectives

The main objective of this work is the development of an architecture for solving the problem of domain-specific image retrieval in distributed environments. To do so, we have two complementary specific objectives.

The first one is to **develop a CBIR architecture based on scientific workflows** in which users have easy access to the underlying techniques. The primary goal of this architecture is to support the definition of a similarity space for image datasets, as this process could greatly benefit from the participation of people who possess domain

[1] <https://photos.google.com>
[2] Source: InfoTrends Worldwide Consumer Photos Captured and Stored, 2013–2017 prepared for Mylio
[3] Source: Annual Imaging and Radiodiagnostics Data by NHS England.

knowledge. However, it is not always the case where these users have enough technical expertise to deal with low-level components of CBIR systems. In this sense, a workflow-based architecture may highly simplify the interaction for end users. The second specific objective is to create **an application interface to that architecture that allows plugging in different data structures specialized in indexing complex data**, making it possible that various metric access methods and algorithms be enhanced with the possibility of working in distributed environments, even without being originally conceived to these scenarios.

Both objectives should be achieved over arbitrary image collections. Scalability is always a concern in CBIR systems as image datasets are already voluminous and increasing their size rapidly. Therefore, we base our approaches on distributed computation which, although more complex, is considered fundamental in such scenarios for performing efficient data processing and searching operations.

1.3 Outline of This Dissertation

This chapter presented an introduction to the theme of this qualification dissertation. The remaining chapters are organized as follows:

- Chapter 2 discusses the fundamental concepts this work relies on and discusses the main related work;
- Chapter 3 presents our contribution for addressing the similarity space definition through a distributed architecture based on scientific workflows and our approaches for employing metric access methods in the distributed architecture for performing efficient similarity search;
- Chapter 4 discusses implementation details, and shows a study case and experiments that evaluate our proposal;
- Chapter 5 presents the conclusion of this work.

2 CONCEPTS AND RELATED WORK

This chapter introduces the fundamentals of content-based image retrieval and similarity search at large-scale, as well as discusses some works proposed to address the challenges related to our objectives.

2.1 Content-Based Image Retrieval

With the increasing amount of images that society produces, we can notice increasing interest in performing Content-Based Image Retrieval (CBIR) over large image databases. The concept of CBIR relates to any technology that assists the management of digital images using their visual content [2].

As Gonzalez e Woods[8] defines, a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are known as pixels and, in its raw form, an image is essentially a pixel matrix with their values being derived from sensors. Sensors are, therefore, responsible for generating images and are part of the first process in a system for image retrieval – *image acquisition*. Noteworthy is that this first step can already influence the semantics of images because the acquisition is usually done with some predefined intent and application, which can demand that a specific sensor is used to generate the images.

Gamma-ray imaging is broadly used in the fields of nuclear medicine and astronomical observations, and X-Ray imaging has found uses in medicine and even industry. That is because these sensing techniques are suitable for capturing details in images that are relevant to their respective domains. On the other hand, more popular techniques of producing images, such as those employed by regular smartphone cameras, are usually based on the visible band of the electromagnetic spectrum. After they are generated, images are typically distributed in some way (e. g. , through the web, or even physically), which is also considered as part of the acquisition process [8].

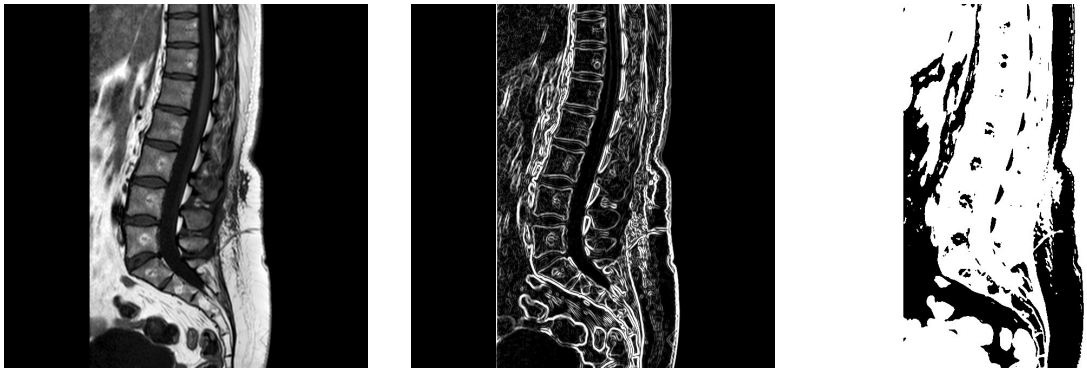
Each of these domains has its own specific set of concerns, and users responsible for analyzing those images are the ones having the knowledge needed to better take advantage of the information each image carries. In other words, that pixel matrix also has semantics, which is dependent upon the user. In the medical field, for instance, observations made by a radiologist about an image (i. e. , its semantics) can even become metadata that is attached to the body scan itself [9]. That metadata could describe what was diagnosed through that exam, for example.

It is not always the case where the relevant visual patterns are easily spotted. Thus a CBIR system frequently relies on treatment of images. This treatment could, for exam-

ple, highlight specific facets of the visual content or make irrelevant details less influential in the whole picture. Based on this, the second process in an image retrieval system is *image enhancement*. Image enhancement can be defined as the process of manipulating an image so that the result is more suitable than the original for a specific application [8]. Usually, this functionality is achieved through image filters – image processing algorithms that receive an image as input and produces an enhanced image as output. Some examples of operations image enhancement filters could perform are: window and level adjustment, edge enhancement, contrast enhancement, noise reduction, binarization, neighborhood-based adjustment, and geometric-spatial transformations, such as rotation, scaling and translation [8, 10].

It is a consensus that it is not possible to conceive a general theory that establishes an algorithm that will perform well for all images [8]. For example, methods that are useful in satellite images probably will perform very poorly if applied to other kinds of images. Image enhancement is also a very domain-oriented step, and a good approach to this process should be guided by the problem one is trying to solve. In the end, the usage of these enhancement techniques aims to achieve the best possible image representation in the context of some application, to be interpreted either by humans or systems [10].

As an example in the medical domain, we have Figure 3a depicting a Magnetic Resonance Imaging (RMI) scan of the spine. That is, it is the original image, the same image generated by the specific sensor (in this case in a clinical environment), and it was not subject to any enhancement filter. Figures 3b and 3c, on the other hand, show the same image after its processing. The image processing algorithms used were, respectively, the Sobel Filter, which is useful for highlighting image contours, and the binarization operation, which maps every pixel to some domain containing only two values (black and white in this case).



(a) Original Image (b) Sobel Filter (c) Binarized

In the context of CBIR systems, the image enhancement step can cooperate with the *feature extraction* one, which is the next process. This step generates a *feature vector* for each image that represents one or more visual patterns. The algorithms that produce the feature vectors are called *feature extractors*, and they usually focus on a specific kind

of image, representing numerically properties based on color, texture or shape patterns, or even a combination of them. The reason for using feature vectors to represent images is that, although a trivial comparison between two images could be performed, based only on their pixels, this operation would not have useful results, since the similarity between images depends more on a high-level description than it depends on pure pixel values [11]. Therefore, several kinds of feature extractors exist, each one with the goal of better capturing features that are relevant for a specific domain problem. Some widespread used feature extractors in the medical domain, for example, based on the works of [10, 9, 12], include gray-level and color histograms [13], edge histograms [14], Gabor [15], Wavelet-based [16], Haralick [17], Zernike Moments [18], Tamura [19], and Fourier-based ones [13].

Image feature vectors are, therefore, complex data types, having the similarity among elements as the most employed concept to organize them. The similarity between two feature vectors is usually computed as the inverse of the *distance* between them in the embedded space. A *distance function* has the form $\delta : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}_+$, where $\mathbb{S}$ is the domain of the feature vector. Therefore, the smaller the distance between feature vectors, the higher the similarity between the images they represent. The combination of the domain of the feature vector and the distance function form a similarity (or metric) space, which works as a mathematical abstraction of similarity [20].

2.2 Similarity Spaces

The main processes presented so far – image acquisition, enhancement, feature extraction, and distance calculation – are some of the main building blocks for the definition of a *Similarity Space*. A similarity space is a metric space in which points represent objects, and the distance between points is inversely proportional to the similarity between the corresponding objects [6]. The variability of algorithms to be used makes similarity evaluation process is usually quite complex and specialized because plenty of approaches is found in the literature. There are a variety of image enhancement algorithms, feature extractors – we mentioned just a few, distance functions [21], and complementary techniques in the literature, such as feature selection [22], which all can work together to achieve a suitable data representation. The objective of using them is to define a *Similarity Space Instance* that best represents the similarity distribution of the target dataset. More specifically, a Similarity Space Instance is a feature vector-distance function pair that may be fine-tuned through operations that affect any of these two components [23]. This similarity space M can be expressed as $M = (D, \delta)$, where D is the data domain (formed by the set of feature vectors), and δ is the distance function. As Figure 4 illustrates, there are several examples of methods that act upon the data itself (the feature vector) and others that act upon the similarity measure (the distance function). Every modification promoted by any of these elements produces a new alternative instance [23].

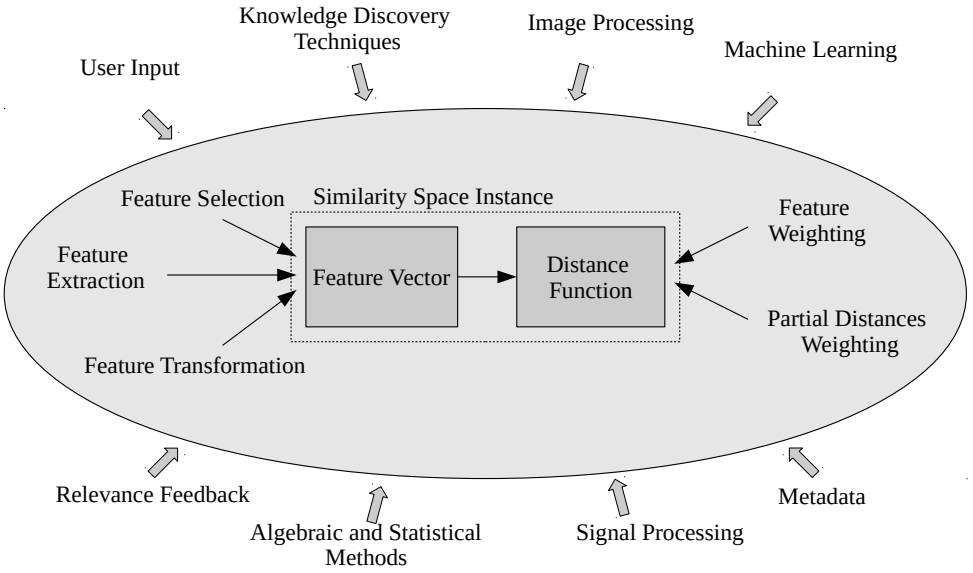


Figure 4 – Techniques involved in the process of defining similarity spaces. Source [23].

Ideally, the defined instance should form a representation of information that is descriptive enough to resemble the perception a human being would have when analyzing them. This requirement imposes a challenge for domain specific applications like medicine. Medical imaging plays a central role in modern healthcare. Some of its most widespread applications include support for diagnosis, treatment planning, and assessing response to treatment. The produced images can also potentially be used as support for evidence-based diagnosis, teaching, and research, by using CBIR methods and tools [3]. The issue of retrieving images by their visual content, including the application of this technique in the medical domain, has been studied for several years [9, 3]. However, defining a proper Similarity Space Instance according to a given purpose is a cumbersome problem.

We refer to the sequence of tasks needed for achieving such a representation as the process of *Similarity Space Definition*. It can be a very complex process that depends on a series of factors. The first one comes from the semantic point-of-view, where the role of a domain expert is fundamental because there can be (and most certainly there are) domain intricacies hidden from an average person when interpreting what information an image contains. Allowing users that possess this knowledge to participate in the process can mean a major step forward in the direction of new and better methods.

However, domain experts participation requires from them a high degree of interaction with many techniques outside from their main expertise, especially with computational tools, such as image libraries, feature extractors, transformations, complex data indexes and search algorithms. There are principled, coordinated and reproducible ways for defining what similarity means in a specific context. They comprehend the ordered application of a set of algorithms, usually forming a pipeline – a sequence of composable and chainable operations. Such a pipeline can be straightforwardly modeled as a Directed

Acyclic Graph (DAG), which is the standard structure used by most Scientific Workflow Management Systems (SWfMS) [24]. Therefore, the process of Similarity Space definition fits in the category of applications that can benefit from the usage of SWfMS.

Scientific workflows are structures intuitive for users to define tasks that involve lots of data dependency and sharing between tasks [24]. They capture the pattern that usually permeates the Similarity Space definition process, offering the advantage of being at a much higher level of abstraction than explicit coding the corresponding function calls and interchanging data structures in a programming language. Thus, using an SWfMS removes a potential barrier end users can have to use complex CBIR techniques and algorithms. Instead, SWfMS usually offer a graphical interface where workflows can be defined and have their execution watched.

There are some works in recent years that provide environments for defining operations on images for analysis purposes. Valente et al.[25] discuss a system that supports the processing of machine learning algorithms on a grid infrastructure for large-scale medical image analysis. Their proposal is also based on workflows, but they are defined programmatically, requiring from their users some familiarity with computer programming. Besides, the system was in earlier stages of development and presented as a very early prototype, and no finished work was found at the time of writing this dissertation.

Following a similar approach, Sridharan et al.[26] introduced PipeBuilder, a framework where users can define pipelines for analyzing large collections of images of human brain. It allows that workflows be refined iteratively by varying parameters and evaluating their impact on the quality of results. The workflow definition is also through programming primitives, using scripts written in the Python programming language. There is a visual interface where users can visualize a graph derived from the provided script and visualize intermediate results to follow the execution status of some sub-task, including images generated after some processing. They show two study cases, one for stroke diagnosis and other for Alzheimer diagnosis. The main limitations of their proposal are: a) pipeline construction is programmatic, something that imposes a barrier for users with no programming experience; b) scalability in PipeBuilder is achieved through an interface with Oracle Grid^{TM}, which limits the possibilities of deployment; and c) their solution is very tied to a specific problem domain, that of brain image analysis, which influenced the framework conception and theoretically could limit its application on other domains other than white matter analysis.

A tendency of more achieving more context-independent scalability concerns can be seen in the VISCERAL project. It was introduced in Hanbury et al.[27], addressing a concern that is shared by many researchers: there is no *de facto* approach when dealing with the comparison of competing methods that seek to improve medical images retrieval in a large scale scenario. They defend a paradigm called Evaluation As A Service (EAAS)

to address that. Some of the issues that are meant to be solved are the very costly data movement through the network, data confidentiality concerns, and frequent changes on datasets. These problems impose a barrier to the proper comparison of new algorithms and techniques from different authors. The volume of data involved in this tasks being so huge, only distributed and parallel environments are capable of storing data and computing complex analytic pipelines. Also, since data movement is so expensive, whenever possible, what should be traveling through the network is only application code and not data. Those are technical concerns we share. Despite having some common goals, VISCERAL's main objective is different from ours. They try to provide an environment where researchers can develop their solutions to predefined problems using public data (at least in the training phase), and that evaluates the different approaches in a fair way. That is very desirable in a competition-like scenario, but not in use cases where users are experimenting mostly with private data. In these cases, there is more exploratory analysis, as users begin a task not being sure about what to find.

Among the recent challenges in similarity analysis of images, the always-increasing size of the datasets can is one of the most critical. Several recent works propose scalable methods for the various steps that are arguably under the umbrella of a standard CBIR system, such as feature extraction [28], clustering [29], indexing and querying [30, 31] and classification [32]. The underlying approaches they take go from map-reduce algorithms – usually based on frameworks like Apache Hadoop [33], Apache Spark [34], Apache Storm [35], or Apache Flink [36] – to peer-to-peer networks [7], solutions that are known to scale well for most problems.

The work of Mera, Batko e Zezula[28] makes a comparative study of how different big data frameworks behave when performing feature extraction from image datasets. They evaluate approaches based on four map-reduce-based frameworks – Apache Hadoop, Apache Storm and Apache Spark – and a solution based on Grid computing infrastructure. As argued by the authors, the feature extraction task can be very tedious, because each object in the dataset needs to be examined independently. However, this means it is possible to adopt an approach with highly parallelized computation, and all the frameworks solutions considered in their work (despite each having their advantages and drawbacks) perform well in problems that have this parallelizable nature. One major drawback the Grid-based solution has in comparison to its competitors is the failure tolerance.

In the experiments performed, the feature extraction algorithm used five descriptors from the MPEG-7 standard [37], implemented on top of the MESSIF library [38], and the dataset consisted of a collection of one million JPEG images collected from the web, which summed about 45. 9 GB of data with no defined domain. Their findings were able to identify scenarios where different alternatives perform better than others. For example, due to its streaming-oriented architecture – when the input dataset is fed to a system in

real-time – the Apache Storm solution is better suited to cases where data should not or could not be stored – *i. e.* , the results are used and immediately discarded. On the other hand, approaches based on Apache Hadoop or Apache Spark are better suited for cases where the input dataset is large (in their experiments, this would be at the count of 500,000 images).

The work of Zhang et al.[4] makes a similar comparison but in the context of an astronomy imaging use case. They compared the performance of image processing pipelines executed on top of Apache Spark with a very domain-specific implementation made in C and running on the NERSC Edison supercomputer. The authors were able of achieving a very competitive performance on the Apache Spark-based approach (within the range of 18.5% slower and 75.1% faster) in comparison to its competitor. However, the Apache Spark solution is much cheaper, not requiring High Performance Computing hardware, and more flexible, as it could be deployed in many environments due to the flexibility provided by the Apache Spark framework.

These two last works are a good basis for stating the major benefits of using big data solutions like Apache Hadoop and Apache Spark in CBIR-related tasks where each piece of data can be processed independently. Those frameworks are well-suited for tasks that operate with all elements of the dataset, like image processing and feature extraction, the first two processes a CBIR system usually performs in the similarity space definition. Nevertheless, approaches based on the map-reduce paradigm are not very adequate when it comes to searching, because of their batch-oriented nature [39]. Parallel and distributed retrieval approaches organize data in such a way just a small piece of the dataset is processed at query time while map-reduce frameworks always traverse the whole input dataset. Doing that in the similarity definition step is a requirement. However, doing that even once in the middle of a search brings many performance penalties, meaning that an answer to a query can be become very slow and latent, which is undesirable in exploratory analysis scenarios, where immediate feedback in queries should be a goal.

2.3 Similarity Queries

Once the similarity space is defined, it is possible to perform similarity queries on it. Similarity queries differ from their traditional counterpart in the sense that the comparison operators usually employed on queries on traditional data (such as $<, >, \leq, \geq$) are not applicable to complex data, and even though the identity operators ($=$ and $\neq$) are applicable, their use is very restrict. For instance, two images that are the same except for a single pixel are not equal, although they are considered visually equal for an end user. Therefore, the most appropriate way of search complex data domain – such as images – is through some distance-based fashion, as similarity queries do.

Similarity queries consider how the elements are distributed in a similarity space. The most common query variations are the Range query and the k-Nearest Neighbor query. A *Range query* (Rq) can be defined as:

$$Rq(\delta, S_q, \xi) = \{s_i \in S | \delta(s_q, s_i) \leq \xi\}.$$

where $S \subseteq \mathbb{S}$ is a dataset on complex data domain $\mathbb{S}$, $s_q \in \mathbb{S}$ is a reference (or query) element, δ is a distance function defined in $\mathbb{S}$, and $\xi \in \mathbb{R}_+$ is a distance threshold. This query retrieves all elements $s_i \in S$ whose distance to the reference element s_q is less or equal to ξ. A *k-Nearest Neighbor query* (kNNq) can be defined as:

$$k\text{NNq}(\delta, S_q, k) = \{s_i \in S | \forall s_j \in S \setminus K, |K| = k, \delta(s_i, s_q) \leq \delta(s_j, S_q)\}.$$

where $k \in \mathbb{N}, k \geq 1$, is the number of elements that should be retrieved, $K \subseteq S$ is the query result set and $|K|$ is its cardinality. This query retrieves the k elements $s_i \in S$ that are the most similar to s_q. Figures 5 and 6 depicts the elements retrieved by examples of the two queries on a two-dimensional space. The Range query of Figure 5 returns the elements inside the ball defined by the query element s_q and the the distance threshold ξ. The k-NN query of Figure 6 returns the $k = 4$ elements that are the closest to s_q, which are shown in figure connected to it by a virtual line.

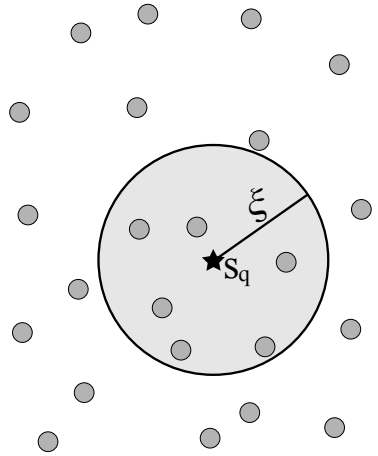


Figure 5 – Range Query

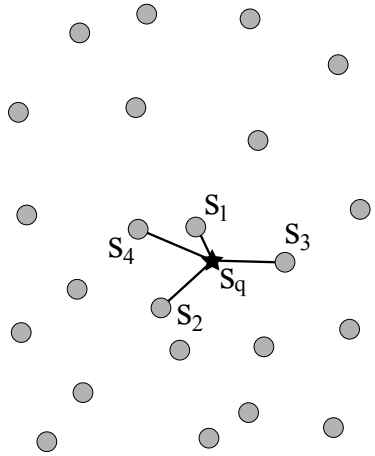


Figure 6 – k-NN Query

2.4 Metric Access Methods

The most time-consuming step in searching by similarity is the distance calculation between pairs of elements, which may require accessing the disk to retrieve the corresponding feature vectors. Therefore, when it comes to searching, the most usual way of speeding up computation is by pruning the dataset in some way, either by diminishing the search space, meaning that a minimal number of objects is accessed when answering a

query, or by diminishing the cost of distance calculation, which means that the domain of the feature vectors should be transformed in some way that makes the evaluation faster (*e.g.*, by eliminating some dimensions through feature selection or by transforming the high-dimensional space into a low-dimension one).

While this kind of optimization is already found in traditional database solutions when dealing with traditional data (as already mentioned, this includes small character strings, numeric data, dates), doing it for complex data domain require the usage of more specialized solutions. Metric Access Methods (MAM) assume that the objects in a dataset are in a metric space and that they can be organized through similarity evaluation.

Several centralized approaches (*i.e.* storage and processing happen on a single machine) that leverage on the metric space concept has been proposed in the last 30 years. The first group of structures consists of static structures, which are structures once created cannot be updated and thus can become inefficient in a very dynamic environment. Examples of this class of structure include the Vantage Point tree [40], Multi-Vantage Point tree [41] and Fixed-Queries tree [42]. Other data structures were proposed to address the dynamism aspect, meaning that they support being updated while still maintaining their structure. Representatives of this group (still centralized) are the M-Tree [43] and one of its successors, the Slim-tree [44]. All these structures divide the search space into regions and provide ways of answering queries without necessarily visiting all those regions. Moreover, they have a very deterministic nature – their answers are always accurate, in the sense that any pruning that is made does not change the answer that would be achieved through a linear (or exhaustive) search. The last class of approaches aims at increasing the efficiency in queries by sacrificing the accuracy – also known as approximate search –, like those based on Locality-Sensitive Hashing (LSH) [45, 46] or in Permutation-based Indexes [47]. Approximate search has been proved to bring very significant performance improvements for applications that do not require exact answers.

These centralized methods were mostly designed for single-node and even single-core environments. The problem is that it is very costly to setup a single machine able to deal with datasets on the big data era. Distribution of processing and storage is known to be very effective in these cases [7]. Since some machines share the responsibility that would otherwise be of only one machine, and the computing demands being the same, this means that (at least theoretically) it is reasonable to expect a performance improvement due to parallelization of tasks. However, if in the case of image processing algorithms and feature extraction the tasks are very straightforward to make parallel – even in multiple-node environments – regarding other tasks, like searching, this is not trivial. Therefore, approaches that specifically target distributed environments for similarity searching complex data are being developed.

For instance, the work of Novak, Batko e Zezula[48] introduced the Metric Index

(or M-Index), a mechanism that brings high performance when searching by similarity. An important feature of the M-Index is that the index and data storage are separated. Thus, it is straightforward to build a distributed version of the index, and the authors did it in a second work by using a Skip graph peer-to-peer network architecture [49]. By being a flexible architecture, it supports both precise and approximate search and allows that the most suitable strategy for a specific context be picked. The index itself uses a fixed set of reference objects (also called pivots) to organize the dataset, with partitioning and mapping schemas being inspired by iDistance [50]. M-Index indexing begins with a set of n pivots from a sample dataset $R \subseteq S$ and applies a Voronoi-like partitioning that divides the space into n disjoint clusters. Each cluster c_i has its own pivot p_i, and a mapping based on the distance of all objects in c_i to p_i is performed to establish their position in the one-dimensional space. After that, each object $o \in D$ has a numeric key assigned according to the distance from its cluster's reference object, which can be used to prune the space in searches.

2.5 Final Remarks

This chapter presented the background concepts important to the understanding of this work and discussed some works that have goals in common with our proposal. In general, there is no single work that has the same scope as ours. We present our contribution in the next chapter.

3 PROPOSAL OF A WORKFLOW-BASED DISTRIBUTED ARCHITECTURE FOR LARGE SCALE CBIR

The main contribution of this work is a distributed architecture for large-scale content-based image retrieval. There are two related, although separate, main problems that should be addressed by an architecture providing such functionality:

1. how to define a similarity space, which would allow users to create searchable representations for image datasets;
2. how to perform efficient searching in the similarity space defined by the user.

This chapter details how the proposed architecture deals with the first requirement, the definition of similarity spaces. Section 3.2 describes the conceptual architecture this work proposes, while Section 4.1 describes the implementation of a prototype adhering to the architecture main ideas. Part of this work work is already published and was presented at the 21st International Database Engineering & Applications Symposium (IDEAS[1]), in 2017 [51].

3.1 Design Goals for Representation and Processing

Similarity Spaces rely heavily on the data domain and on how users perceive it. For example, the semantics of an image in the medical domain differs a lot from those of an agricultural one. Moreover, even when restricted to the same image domain, the perception users have depends on the specific task they are facing, as well with the previous knowledge they have. As an example, the visual patterns a radiologist would look for in a Computerized Tomography (CT) scan of the chest when diagnosing some interstitial lung disease are very different from those he/she would look for in a Magnetic Resonance Imaging (MRI) of the spine when diagnosing a spine fracture, although they are both in the medical domain. That means that the concept of similarity between data objects is subject to huge variations, according to the application context.

The aforementioned particularities translate into the requirement that a system for defining similarity spaces should provide a high degree of flexibility regarding what operations users can perform (*i.e.* how expressive the underlying system is), how easily they can do that (*i.e.* how intuitive is the interface where they interact with system) and how fast the operations are executed over large datasets. Therefore, *our contribution is an architecture for supporting large-scale CBIR that adheres to three main design goals:*

[1] <http://confsys.encs.concordia.ca/IDEAS/ideas17/ideas17.php>

workflow-based user interaction, expressivity and scalability. We discuss these design goals in the rest of this section.

3.1.1 Workflow-based User Interaction

One of our goals is to ease the experience of users in embedding their domain knowledge in image retrieval activities, and this influences how we approach the problem. As users could interact with a very large number of algorithms to design a task, it is important that a robust way of describing tasks be available to them. Our proposal employs Scientific Workflows Management Systems (SWfMS) for modeling and coordinating the execution of tasks involved in similarity space definition, for several reasons:

- SWfMS allow users to focus only on the problem description in a declarative way, without worrying about details in the implementation level (the user should know *what* an algorithm does to the point of being able to use it, but not necessarily know *how* to implement that algorithm);
- the graph-like representation means that algorithms can be chained through dragging and dropping graphical box-like components, which is very intuitive for non-technical users and has been considered a good fit for problems that have dataflow characteristics [52];
- most SWfMS provide ways of logging the tasks performed in detail, including the algorithms and parameters used, which makes possible for workflow sharing and even mining [53]. This allows that successful solutions be replicated or adapted to similar problems in a collaborative environment;
- the description of a workflow is decoupled from its execution; this means that users can use thin clients to interact with systems that are physically located somewhere else, including in cloud environments.

In general, each task (or node) the user adds to a workflow represents a stage by which data travels, and the link between them represents the direction data flows. How to build a workflow in practice depends on the task at hand.

3.1.2 Expressivity

Expressivity is often needed in CBIR systems because the problem users are trying to solve is not always fully known *a priori*, and the iterative nature of the similarity spaces definition can influence the process as they discover new information. Therefore, our architecture includes a minimal set of primitive operators known to be suitable for defining similarity spaces for image datasets. These operators should have a very composable

nature, allowing users to chain an arbitrary number of tasks, as long as there is an agreement between the codomain of one operation and the domain of its successor. From these building blocks, users can then build more complex tasks that correspond to whole image retrieval tasks.

Some of the most common operators were discussed in Chapter 2, and their main role is to process data in the various stages through which it goes. In the case of an image dataset, some of the capabilities those operators should implement include:

- basic methods for image reading and writing, including the ability to deal with domain-specific image formats, such as the DICOM standard in medicine;
- image processing algorithms, such as enhancement and noise reduction filters, and other techniques that highlight specific aspects of images and improve feature extractors ability of properly representing their visual content;
- algorithms for feature extraction of various flavors, covering the most common patterns found in images, such as color, shape, texture, and their combination;
- approaches for querying data by similarity;
- result analysis tools that allow the evaluation of a processing pipeline suitability for a given task.

This set of complementary operations are available to the user, who is responsible for defining a workflow that includes the source dataset, the steps in the processing pipeline, and what kind of output (if any) he or she wants the system to produce. Multiple pipelines could also be combined in the same workflow, in order to reuse intermediate results of a operation and vary the subsequent operations that process data or their parameters, mainly to compare how this variation impacts retrieval quality and/or performance. As an example, Figure 7 shows an abstract workflow that combine multiple pipelines in one workflow.

3.1.3 Scalability

The last main design issue is that users will often face the need of dealing with large volumes of data. Image datasets are increasing both in the number of elements they contain and in the size of those elements, and new CBIR systems should be designed to fulfill this demand. Furthermore, in an age when even personal workstations almost always have multi-core processors, it is wasteful not to seek opportunities for using all the computational resources available.

Fortunately, there are some kinds of tasks that are well-suited for parallelization, as is the case of most of the operations involved in the similarity space definition step. The

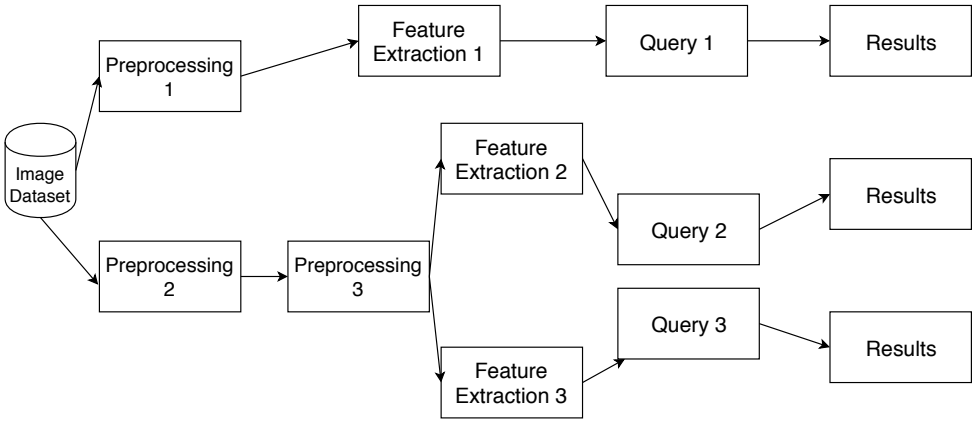


Figure 7 – Multiple Pipelines Combined into one Workflow

definition of similarity spaces contains some data-parallel operations – operations that can be performed to each data object independently – such as reading and writing data to persistent storage, and applying image processing and feature extraction algorithms to individual images.

A non-distributed (*i.e.*, single node) architecture could already benefit from this data-parallel reality by delegating the responsibility of processing different pieces of data to different processing cores. By doing that, a system can roughly apply as much parallelization as there are processing cores dedicated to the task, which reduces processing time proportionally. In a distributed (*i.e.*, multiple nodes) setting, the situation is actually similar. The processing of each piece of data can be delegated to different computing nodes, which could further split data into smaller pieces to be processed by each of its processing cores. The difference in the distributed case is the fact that there are extra concerns regarding cluster management, load balancing, data distribution among nodes, fault tolerance and failure recovery.

We argue that is beneficial for an CBIR architecture to be made on top of a framework that addresses these common requirements to distributed systems. These frameworks offer abstractions over distributed processing and storage, taking advantage of the parallelization opportunities and making it feasible to deal with datasets that cannot be efficiently handled by centralized approaches. It is noteworthy that these frameworks do not require using any hardware specialized for High Performance Computing. In fact, they were designed to run on top of commodity hardware, prioritizing horizontal scaling (which increases processing power by adding new machines) over vertical scaling (which increases processing power by adding more resources (RAM, CPU) to an existing machine). This makes them an accessible and appropriate solution for being used as the basis for this step in the image retrieval architecture.

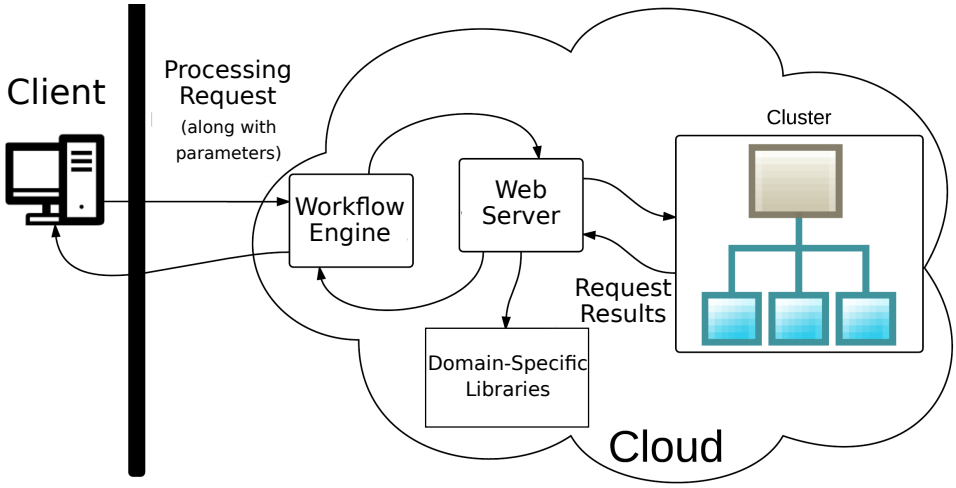


Figure 8 – Interaction of the Components in the Proposed Architecture

3.2 The Proposed Conceptual Architecture

Considering the three points mentioned above (workflow-based user interaction, expressivity and scalability), a conceptual architecture is drawn in Figure 8. We developed an architecture for large-scale image retrieval that can be orchestrated through scientific workflows. The usage of workflows lowers the barrier for domain experts to participate in the process of defining similarity spaces by hiding details like execution model and computational environment setup. This workflow-based approach allows for the employment of cloud-based high-performance solutions, which makes it possible that processes specified by users operate even in big data sized datasets. The figures depicts a high-level view of how the different components of the architecture interact with each other in the execution of a generic task.

The starting point of a task comes from a thin client machine (on the left side of the figure), where the user designs a workflow that contains a sequence of sub-tasks whose execution creates a similarity space for the input dataset. The usage of a thin client frees the user from having any special hardware for performing the experiments. The workflow defined is then validated both syntactically (guaranteeing that the workflow tasks have valid inputs and outputs) and semantically (such as data types between adjacent operations being correct). These validations occurs on the workflow engine, which may be allocated at a dedicated server or embedded in the workflow design interface – this depends on which workflow solution is used. If any errors are committed in the workflow design, they are immediately reported to the user, before any task execution on the cluster is triggered. The validated workflow is then compressed (using some XML-like specification format, for instance) and sent to be executed by the hardware responsible for the actual execution (there are some workflow systems that have the workflow execution engine integrated into the definition interface, but they can be considered as independent components).

The workflow execution server is then responsible for mapping the tasks defined in the workflow to web service calls that invoke domain-specific functionality (*i.e.* the algorithms for image processing, feature extraction, and input/output). The point of using web services is that they allow remote invocation of applications that are able of process potentially huge datasets by using dedicated environments, such as the ones offered by cloud providers. These web services calls are received by another server and interpreted according to the execution context. Basically, every action contained in the initial workflow has a corresponding algorithm and this step is where the mapping between one and the other happens. Since scalability is a major concern, it is the responsibility of the system to delegate all the heavy processing to a cluster of machines, speeding up computation to give the user an answer in a reasonable time, even for large datasets. After this stage, which may be composed of several sub-stages, the results required by the client can either travel through the same path, in a stack-like fashion, until they reach the client machine, or be stored on the cluster being used in future tasks.

This scheme assumes that the dataset is located in the cluster before any computation takes place. If that is not the case, it is possible to specify another task that downloads the data from some remote location and stores it in the cluster of machines. Although data could also flow between the cluster and the client at the time of workflow execution, the general use of the proposal is that traffic between both ends should be kept to a minimum. It is desirable that what will travel through the network consists only of commands, processing requests, and their parameters, instead of data. This brings the computation to where data is located – a design choice that can improve performance up to orders of magnitude.

This way, the user can define experiments, evaluate their outputs and, in an iterative way, change algorithms and their parameters, always benefiting from the computational power of several machines to perform the heavy tasks.

3.3 Distributed Processing

The distributed processing in our architecture uses the framework Spark [34], however other related technologies that implement the map-reduce model could be employed. The following sections describe the general execution patterns for image processing and feature-based operation tasks as well as for query-based tasks.

3.3.1 Image Processing and Feature-based Operations

With regard to the processing tasks to define the similarity space, we consider two main data types. The first one is the image type, which can represent images from the input datasets as well as modified images after processing tasks that generate images as

output. The second data type is the feature vector type, which stores numeric features produced by feature extraction, selection, selection and fusion tasks. Feature vectors are the input for the query-based tasks, employed for retrieval and analysis operations.

Figure 9 shows what happens in the architecture from the point-of-view of the data, mainly how data is stored and processed, regarding the workflow execution of an example of a simple feature extraction. The input to this kind of task is a dataset consisting of images, which was downloaded and stored in the cluster before the task started. Although the dataset is viewed as a unit by the application, the distributed filesystem in conjunction with the processing framework (in our proposal, HDFS and Spark, respectively) actually divides data into several partitions, and one or more partitions are assigned to each of the cluster nodes. Therefore, each node is responsible for processing the partitions it contains.

Following the scheme shown in Figure 8, the workflow created on the client machine is sent to a server (Master Node) that is responsible for mapping each workflow activity to an algorithm that is implemented in the framework, and then making the appropriate calls that will process the dataset. The processing algorithms are sent to each one of the nodes, that will apply the necessary transformations to the partitions of data it possesses. When computation is done, the resulting dataset (consisting of feature vectors in this case) is written to persistent storage for further usage. Other processing tasks regarding image processing and feature extraction and further transformations are processed in a similar way.

Until now we covered the part of the architecture that is responsible for the similarity space definition by the user. By using it, a non-technical (and possibly domain expert) user can create workflows describing processing pipelines for creating representations of a image dataset that enables similarity search techniques to be employed. The possibility of leveraging on distributed processing solutions ensures that workflow executions can be performed in a feasible time, even when datasets are voluminous.

3.3.2 Distributed Similarity Search

The execution of a similarity space definition workflow for a image dataset, in the context of this work, will usually result in a dataset consisting of feature vectors, where each vector corresponds to one image from the original dataset. This set of feature vectors is the representation of the data where similarity queries can actually be performed. We will now describe how our architecture deals with the actual data retrieval on already defined similarity spaces.

As was the case in the feature extraction tasks, the starting point of a retrieval task is the request created by the user in a client machine, through the construction of a workflow. This request should contain parameters that are specific to a similarity query.

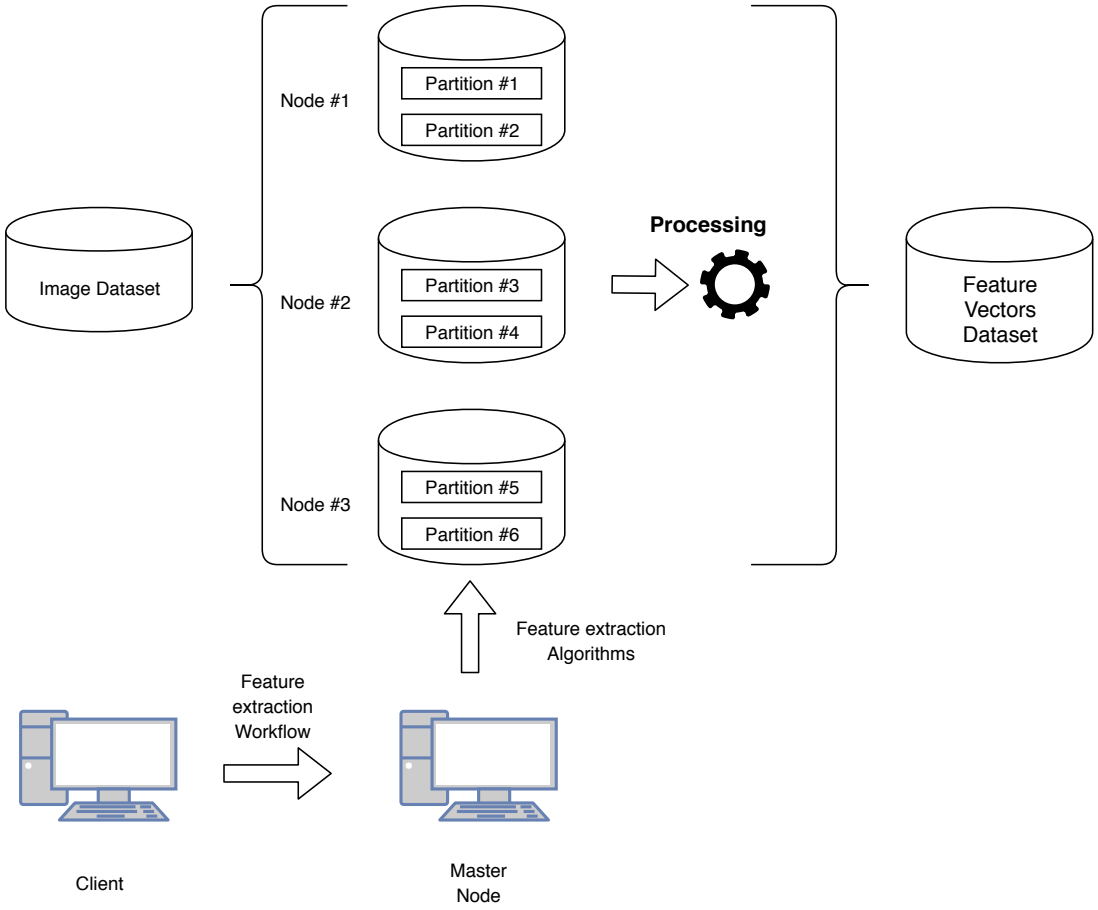


Figure 9 – Data organization and processing in a feature extraction task

The main challenge for distributed similarity search is to decide the processing approach to be used. Existing works regarding distributed search focus on the development of specific access methods in a distributed environment. These approaches are efficient, however they lack flexibility and the environment configuration is specific, which make very difficult to embed them in end-user domain applications. On the other hand, big data distributed processing environments allows very flexible configurations, while hiding from the user distributed processing intricacy, and have been increasingly supported by application development languages and frameworks. However, their general execution pattern is guided to process the whole input dataset at once, which contrasts to the search execution pattern that iteratively prunes the search space to improve query performance.

In the recent years of research in similarity search, several Metric Access Methods (MAM) have been developed with the intent of speeding up similarity queries performance. One of the most time consuming steps in these queries is related to accessing elements in persistent storage, in particular to perform distance calculations. Therefore, one of the ways MAM achieve performance boosts in similarity queries is by ensuring that disk accesses happen only when it is really necessary (*i.e.* for objects that may actually be part of the answer to a query). The actual strategies and policies for doing that vary among different MAM. However, many of these MAM were not conceived to be used on

distributed settings, and even if they reduce search time and costs by a great margin, they mostly operate only as centralized data structures. Therefore, we included in the architecture two main approaches for similarity query execution: (i) a distributed sequential search, which operates over the whole dataset; and (ii) a plugin-based search, which allows including any access method "as is" to the distributed environment. Both of them can be used in the same environment, but they differ in how the queries are processed. The next sections detail how these two search approaches work.

3.3.2.1 Distributed Sequential Search

The distributed sequential way of performing Range and k-NN queries inside our architecture follows the scheme in Figure 10. In the case of a Range Query, the parameters include the dataset to be used, a reference element Sq (*i.e.* reference image), and the value of ϵ, the distance threshold of the query. As discussed in Chapter 2, the answer to the query will include all elements in the dataset whose distance to the reference element is a value less or equal to the distance threshold. In a k-NN Query, the parameters the user should provide are, again, the dataset to be used, a reference element Sq, and the value of neighbors k. The answer to this query will include the k elements with the smallest distance to Sq.

In the figure, the dataset consisting of feature vectors is stored in the cluster of machines following a partitioning scheme that ensures that each node contains an appropriate number of partitions. The number of partitions is calculated based on the hardware available, ensuring that resources are used evenly across the cluster, avoiding both performance bottlenecks and underutilization of resources.

The Master Node receives the request with its parameters from the client workflow and transmits them to each node in the cluster. For both types of queries, each cluster node will iterate through the elements in its partitions and compute their distance to the reference element. A single partition p is traversed sequentially, but since the dataset is divided into several partitions, multiple partitions will be traversed simultaneously.

This first stage will produce, for each partition, a partial answer. If the query requested is a range query, the partial answer is the set of elements in that partition whose distance to the reference element is less or equal to the distance threshold, and the final answer is simply the union of these sets of partial answers. Therefore, the Master Node would simply send to the client machine a set containing all these elements.

If the query is a k-NN query, each partition traversing will return as partial answer the k objects in that partition that are the closest to the reference object. In order to a final answer to be achieved, the k elements from each partition p are sent to the Master Node, that will be responsible to filter which elements are the actual k nearest neighbors in the whole dataset. After this filtering step, the Master Node sends the answer back to

the client, where the user can access the answer to his request.

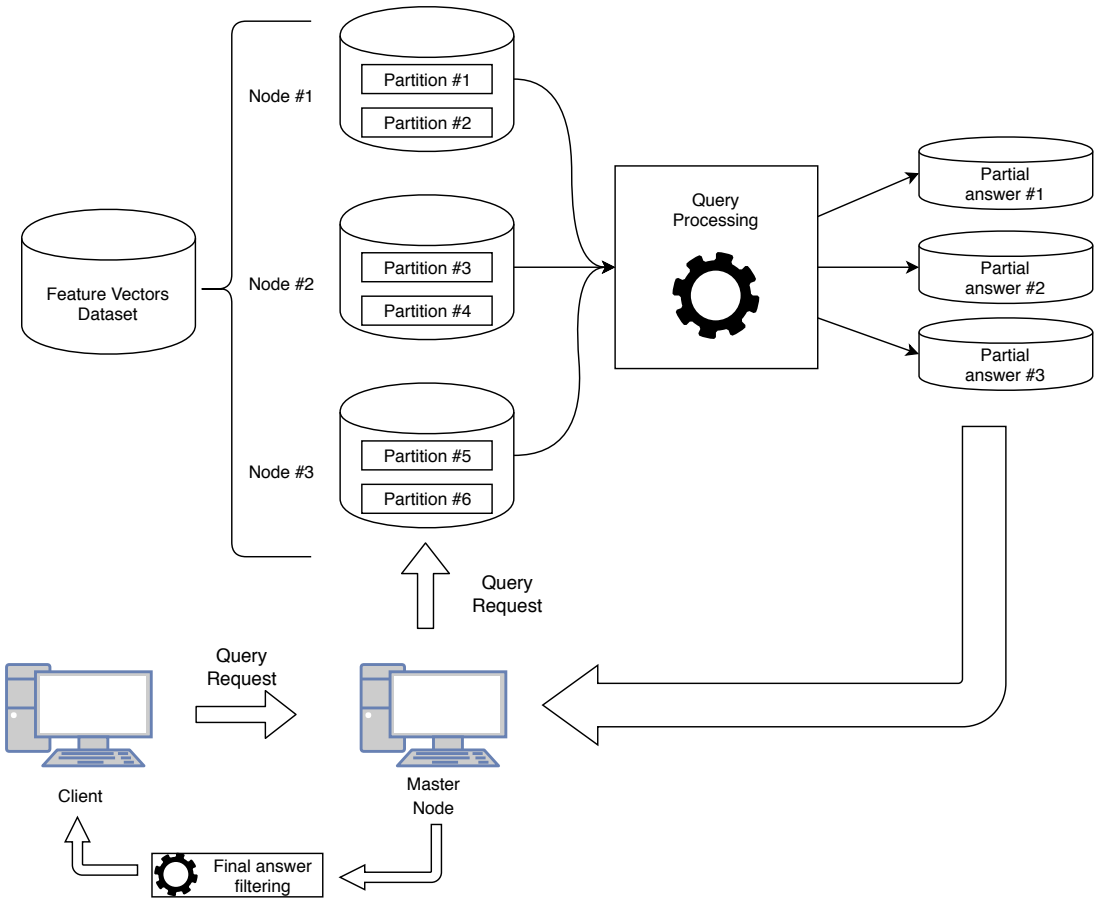


Figure 10 – Query Processing in the Distributed Sequential Search

3.3.2.2 Distributed Search Space Pruning Through MAM plugin

The second approach for searching in our architecture involves enabling the usage of this kind of MAM as a plugin to the distributed architecture, without any modification to their original source code. By doing this, despite employing these solutions in contexts where they were not designed to operate, it is still possible to benefit from their capabilities of reducing disk accesses and achieve performance increases when performing distributed queries. The ability to reuse implementations that might have their use limited due to their inability to keep up with a multinode and multicore scenario is also a benefit of this approach.

Since this search approach is more elaborate than the distributed sequential search, it is divided into two main stages. The first one, when the index is built, can be seen in Figure 11. The process begins in a way very similar to the sequential approach, with a request being sent from the client machine to the Master Node. Parameters in this request include the dataset that should be indexed and the MAM to be used. The initial organization of the dataset consisting of feature vectors in partitions distributed across the cluster nodes is the same of the previous approach.

Each of these partitions is used as input to the index creation sub-task, invoked as a process external in the node's host operating system. This means that the elements of that partition are streamed, one by one, to a process that is running the code of the specified MAM in its own application context. How the MAM actually process the elements is not known to the rest of the application, that treats it as a black box. The only requirement is that each partition be mapped to one for a eassingle persistent data structure (*i.e.* one that is written to the disk), and that the path to the index created be returned as the answer of the task. The completion of this task means that all partitions on the dataset are indexed and all the indexes paths are known to the Master Node.

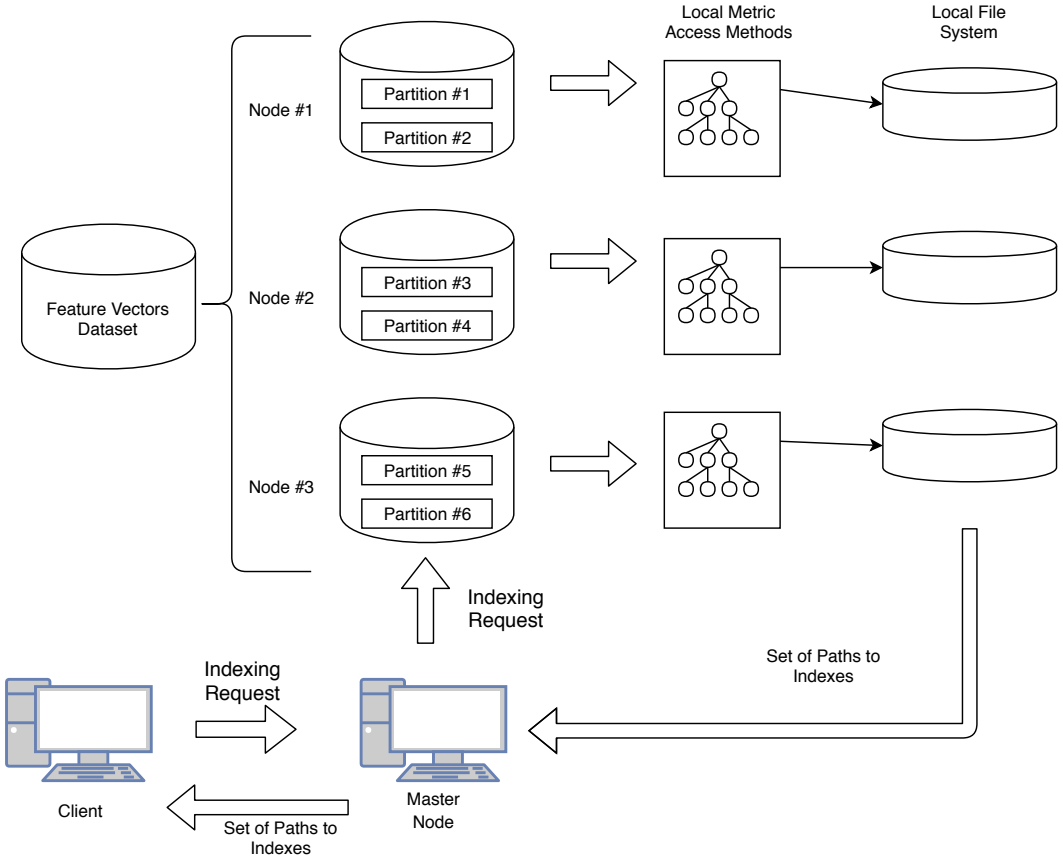


Figure 11 – Indexing in the plugin search approach

After the index construction stage, the system is ready to receive query requests that will use the indexes just created. Figure 12 illustrates the case when a user makes a request to query the dataset using the MAM plugin approach. Parameters to this request include the regular query parameters – reference element Sq and the query radius for Range queries or a k value for k-NN queries – and the location of the indexes created in the last step. The Master Node receives the request and forward it to the cluster nodes. Each node contains one or more indexes that are then used in the query evaluation process and, for each index, a different process is invoked in that node. Performance characteristics, pruning capabilities and distance calculation strategies are the same way

that the used MAM is supposed to have. The rest of the query processing is similar to the sequential approach, in that each index produces a partial answer that is sent to the Master Node, that is responsible for the necessary filtering, according to the type of query requested, and then to send the final answer back to the user who requested it.

It should be noted that since the two stages (indexing and querying) are separated, it is not necessary that a index is created on each query request, as the indexes are automatically persisted on disk and can be used by any number of subsequent queries. On the other hand, if no index is created, then the only to perform queries and retrieve data is through distributed sequential search.

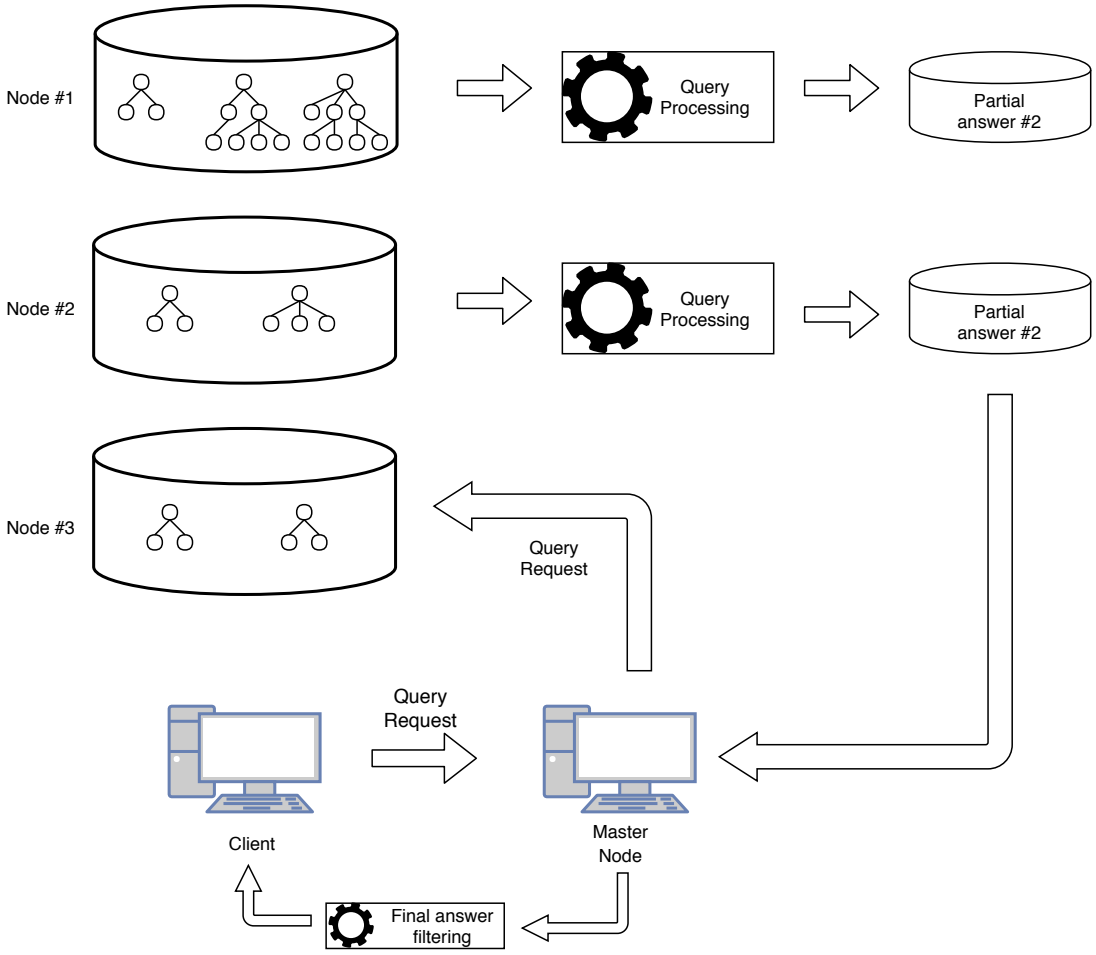


Figure 12 – Query processing in the plugin search approach

Despite having a similar structure, the main difference between the two searching approaches relies on the plugged MAM capability of disk access when evaluating a query. In the sequential approach, even though there is parallelization of the evaluation, all elements of the dataset end up being inspected at some point and therefore the processing time for answering a query is linearly correlated to the size of the dataset. This is mainly a characteristic of the underlying mapreduce framework – they are optimized for tasks that necessarily involve accessing all elements of a dataset, which penalizes operations that need to use only parts of the dataset. In the plugin approach, the time will vary

according to which MAM is being used, but it will almost always be sub-linear due to search space pruning.

4 IMPLEMENTATION AND EVALUATION OF THE ARCHITECTURE

The description of the architecture on a conceptual level still leaves room to some design decisions that should be addressed when creating an actual instance of the presented ideas. In this chapter we describe our instance of the architecture, including implementation details, how it works in general, and show some CBIR tasks it supports and how they can be evaluated. This chapter is organized as follows: Section 4.1 presents the technologies we used to develop our prototype and what is the function of each one of them. Section 4.2 shows a case study of a common task in CBIR systems and how our prototype solves it. This case study involves the processing of large sets of images and feature extraction in distributed environments, In Section 4.3, we discuss some performance characteristics of the tasks we executed to test our prototype.

4.1 Prototype Implementation Overview

The architecture described in Chapter 3 is decoupled of any specific choice of technology, but to evaluate how is its behavior in real scenarios, we developed a prototype implementing the ideas presented. This section gives details about this prototype.

Figure 13 shows how the libraries and frameworks employed are organized in a logical way. These are all open source projects that were properly configured in order to work together.

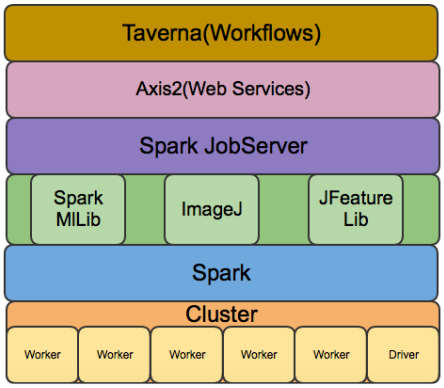


Figure 13 – Technology Stack of the Prototype

Beginning at the top layer, the closest to the user, it is the Taverna Workbench[1], which we use as an environment for both workflow definition and execution. We made this choice because of its graphical interface for workflow definition and its support for

[1] http://www.taverna.org.uk/

invoking remote web services. The Taverna project is one of the main open source solutions regarding Scientific Workflow Management at the time of writing, and our decision to use it is because it supports basic web service interaction. This is because our architecture is based on the remote invocation of costly algorithms that will process loads of data – done through web services – and while there are many other competing workflow solutions, many require the data to pass through the client machine, which is unfeasible for the datasets we aim for. Workflow activities execution consists of web services calls, for which we used the Axis2[2] Web Services engine. In particular, we take advantage of its WSDL-based interface, the same one the Taverna system was configured to use. The calls Axis2 receives from Taverna carry information about which operators it should invoke and their respective parameters. The Axis2 engine is then responsible for triggering RESTful requests in Spark JobServer[3], a system that enables task management in Apache Spark through a REST interface.

The Apache Spark framework[4] receives from the Spark JobServer all the information it needs to process data according to what the user specified, including what dataset to use and which algorithms to apply to it. The set of algorithms for image processing comes from ImageJ [54] and the feature extractors from JFeatureLib [55]. Spark is in charge of managing cluster resources and providing fault-tolerance at the application level. Finally, we use the Hadoop Distributed File System (HDFS) for distributed fault-tolerant data storage. The implementation also contains code that integrates all these libraries with each other, along with utilities that complement them with the features the proposed architecture needs.

The implementation of both search approaches presented on Chapter 3 rely primarily on Apache Spark's capacity of coordinating processes on multiple machines at once and on how HDFS handles data partitioning. Since each node "owns" part of the dataset, searching is a matter of issuing of telling how nodes should inspect its elements and returning results according to the parameters of the query.

The sequential approach utilized the standard data structures from Apache Spark, specially the Resilient Distributed Datasets (RDD). This in-memory data structure consists of a set of partitions, where each partition is an array of elements and belongs to only one node. Therefore, RDD partitions are iterated sequentially by the node it belongs to and relevant elements are filtered as part of the query answer.

Initial tests with the plugin approach use the Arboretum library[5], specifically the MAM Slim-tree [44]. The Slim-tree implementation is in C++ and is the same originally published by its authors. Since it is several years old, it was conceived to be ran in a single

[2] http://axis.apache.org/axis2/java/core/index.html
[3] https://github.com/spark-jobserver/spark-jobserver
[4] http://spark.apache.org/
[5] <http://gbdi.icmc.usp.br/arboretum>

node, single threaded. In this kind of environment, and for smaller datasets, it performs well. However it should be noted that this single-threaded MAM as is could not handle indexing the large datasets used in our experiments.

In our architecture, an application layer integrates the Arboretum library and it is responsible for invoking Slim-tree functionality as a Spark subprocess. This layer invokes Arboretum-specific code in each node of the cluster, and creates a different process in the operating system for each processing core the nodes have, both in the index construction and in the query stages. In practice, what this accomplishes is that multiple Slim-trees coexist simultaneously and receive the same commands when a query is issued, although each one of them operate in distinct parts of the dataset. Each Slim-tree does not know about the others and, therefore, builds indexes and process queries in the same way it was intended to.

4.2 Case Study for Similarity Space Definition

This section demonstrates how the proposed architecture can be used to define similarity spaces in a very specific scenario. As an illustration of the expressiveness of the architecture proposed in this work in a problem that requires a similarity space definition, we reproduce a simplified version of an algorithm for lung disease analysis.

There is a vast literature exploring new CBIR approaches employing image processing techniques for aiding lung diagnosis. The work of Zayed e Elnemr[56], for instance, presents a pipeline consisting of three image processing algorithms: the first one for noise reduction, the second for contrast enhancement and a final one for segmenting lung regions. After segmentation, the authors extract Haralick features from lung regions and make an analysis of its suitability for detecting several lung abnormalities.

The sequence of operations in this pipeline proposed by the authors is applied equally to all images in the input dataset, and a way to model it as a Taverna Workflow with the operators in our architecture can be seen in Figure 14.

The input for this workflow is a dataset consisting of lung images (passed as a parameter in the `dataset_uri` activity). Each of the subsequent workflow activities then corresponds to a step in the pipeline of the authors' technique (`histogramEqualization`, `medianFilter`, and `haralickFeatures`), or to sub-tasks specific to the architecture that are related to data input or output (`dicomLoader` and `write_features_to_hdfs`). If any activity requires parameter adjustment (as the `medianFilter` requires a pixel radius to be specified, for instance), the user can expand the corresponding activity and enter the value he or she desires (default values are used when a value is not provided).

After finishing the task that defined the similarity space, the set of feature vectors is written to persistent storage (the path where the system writes it to is specified in the

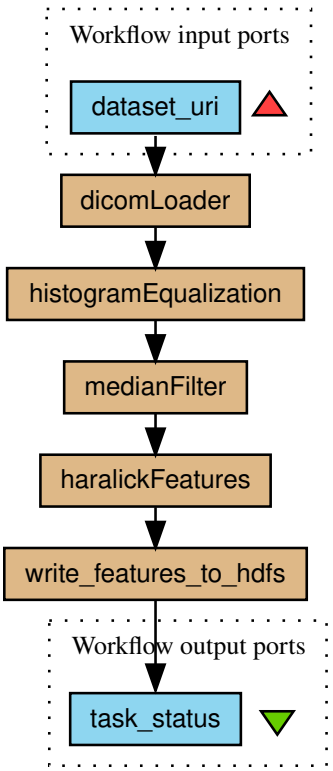


Figure 14 – Workflow for Feature Extraction of LIDC dataset

`write_features_to_hdfs` activity) and is available to be used for similarity queries in the future. At the end of the workflow, the user receives feedback of whether the execution was successful or not.

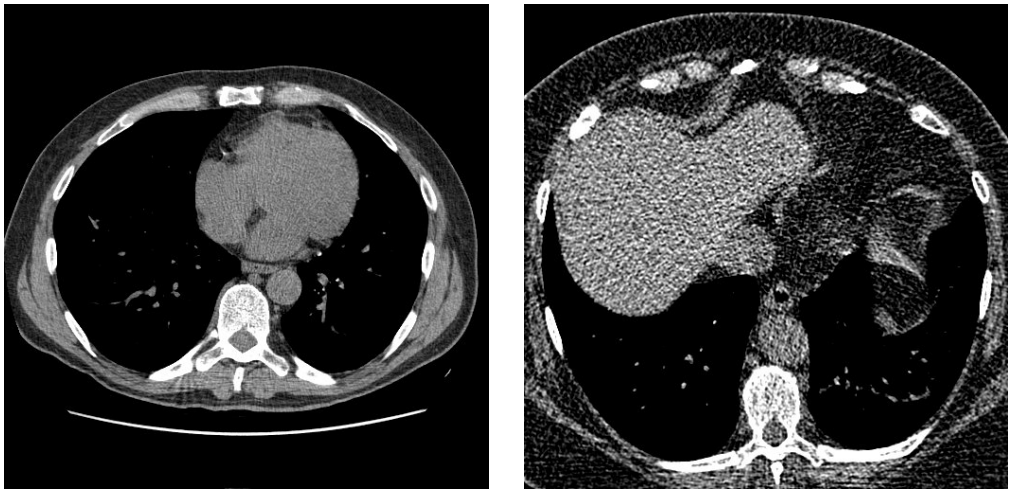
We tested this workflow with a real dataset to evaluate some aspects of our architecture. For this case study, we make use of the Lung Image Database Consortium image collection (LIDC-IDRI)[6] [57, 58], which consists of lung cancer screening thoracic computed tomography (CT) scans with marked-up annotated lesions. It contains 244,527 images and a total size of 124 GB. Figures 15a and 15b show two examples of lung CT scans present in the LIDC-IDRI dataset.

4.3 Performance Evaluation

In this section, we describe the experiments performed to evaluate the performance of feature extraction and search tasks in a prototype that adheres to the architecture we presented. Here, we are concerned about the behavior of the various components of the architecture in a scenario where there are clear opportunities for parallelization, with the goal of assessing how much speed up occurs in a task when different setups are employed.

There are two main tasks that we evaluate. The first one is a **feature extraction**

[6] Available at: <http://doi.org/10.7937/K9/LIDC.2015.LO9QL9SX>



(a) (b)

Figure 15 – Examples of two CT scans in the LIDC-IDRI dataset

task that takes as input a set of images and produces as output a set of feature vectors. The workflow for this task was described in Section 4.2. We employ the LIDC-IDRI dataset for evaluating performance on the feature extraction task.

The second task is one for making similarity queries in an already extracted set of feature vectors. The dataset used for evaluating query performance is the CoPhIR collection[7] [59], which is extracted from the Flickr archive, and it consists of image descriptors and metadata from 106 million images. The CoPhIR collection was built primarily with the purpose of being used in the scalability evaluation in similarity search methods. When performing our experiments, we consider only the five MPEG-7 descriptors (or features) provided for each image (no metadata were used). The descriptors are Scalable Color, with 64 dimensions, Color Layout, with 12 dimensions, Color Structure, with 64 dimensions, Edge Histogram, with 80 dimensions and Homogeneous Texture, with 62 dimensions. The five descriptors have 282 dimensions when combined, and we make use of all of them in our experiments. We employed the Euclidean distance for querying both datasets.

All the experiments presented here were carried out on a cluster consisting of seven machines, one dedicated to managing resources and task distribution (the master node) and six for actual processing (the slave nodes – when not using the full cluster we explicitly point that out in the corresponding experiment). All nodes were connected by a 100M Local Area Network and had a similar amount of resources dedicated to running the experiments: 4 processing cores (Intel®Core™i5-2400 processor), 3GB DDR3 1333MHz of RAM, 250 GB of storage with a factor of replication of 3 in the Hadoop Distributed File System version 2.6.0. Apache Spark version was 2.1.0, Java Virtual Machine version was 1.8.0_121, Taverna Workbench version was 2.5, Axis2 engine version was 1.7.0, JFeatureLib version was 1.6.5 and ImageJ version was 1.4.6. Data is always evenly distributed

[7] Available at <http://cophir.isti.cnr.it/>

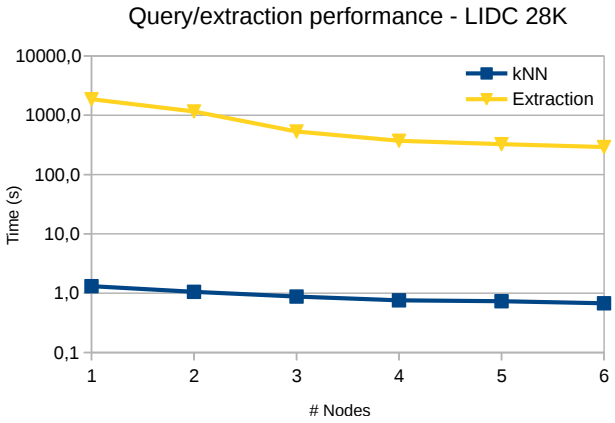
through the cluster, with rounds of load balancing being automatically executed when any given pair of data nodes differ their block usage percentage by a margin higher than 10%.

4.4 Results and Discussion

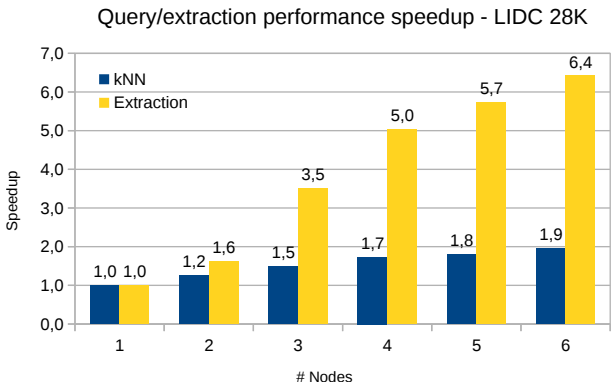
We employed the workflow of Figure 14 to evaluate the performance of our approach for feature extraction. Regarding query tasks, we run k-NN queries such that the 50 query centers randomly selected by a method based on Bernoulli sampling. For each center, we query its 100 nearest neighbors ($k = 100$), using the Euclidean distance metric. The reason for evaluating only k-NN queries is that they can be a bit more complex when performed on a distributed environment, since they require a final aggregation step in the master node, after all the worker nodes having produced their partial answers. On the other hand, for range queries, all the partial answers in the worker nodes would already be part of the final answer.

We report the average time needed for the master node to get to the final answer, after the aggregation step. This same master node is then in charge of passing the query answer to the user, as described in our scheme.

Figure 16a shows that the extraction task in LIDC-IDRI dataset presented an improvement with the increase in the number of processing nodes much higher than query execution. This is due to the fact that this kind of pipeline has the characteristic of being highly parallel, since each piece of data is subject to simple transformations, and there is no aggregation step involved. Tasks of this type can almost always get linear scalability when more computational resources are made available. In fact, Figure 16b shows that extraction achieved a speedup (the proportion of execution time and workload) superior to the number of nodes with three nodes or more while the speedup of query execution was always below 2.



(a) Average time for queries and extractions



(b) Speedup for increasing number of nodes

Figure 16 – Query execution and extraction performance for LIDC-IDRI medical dataset

We also performed queries in a much larger dataset, in order to assess query execution in a more complex scenario. For this, we employed the CoPhIR dataset. Firstly, we evaluate the system's performance when using the sequential approach for searching, noting how system responds when the input dataset contains 1, 10 or 100 million elements. For each of these cardinalities, we also vary the number of nodes responsible for answering the queries. Figures 17a to 17c report query performance under these conditions.

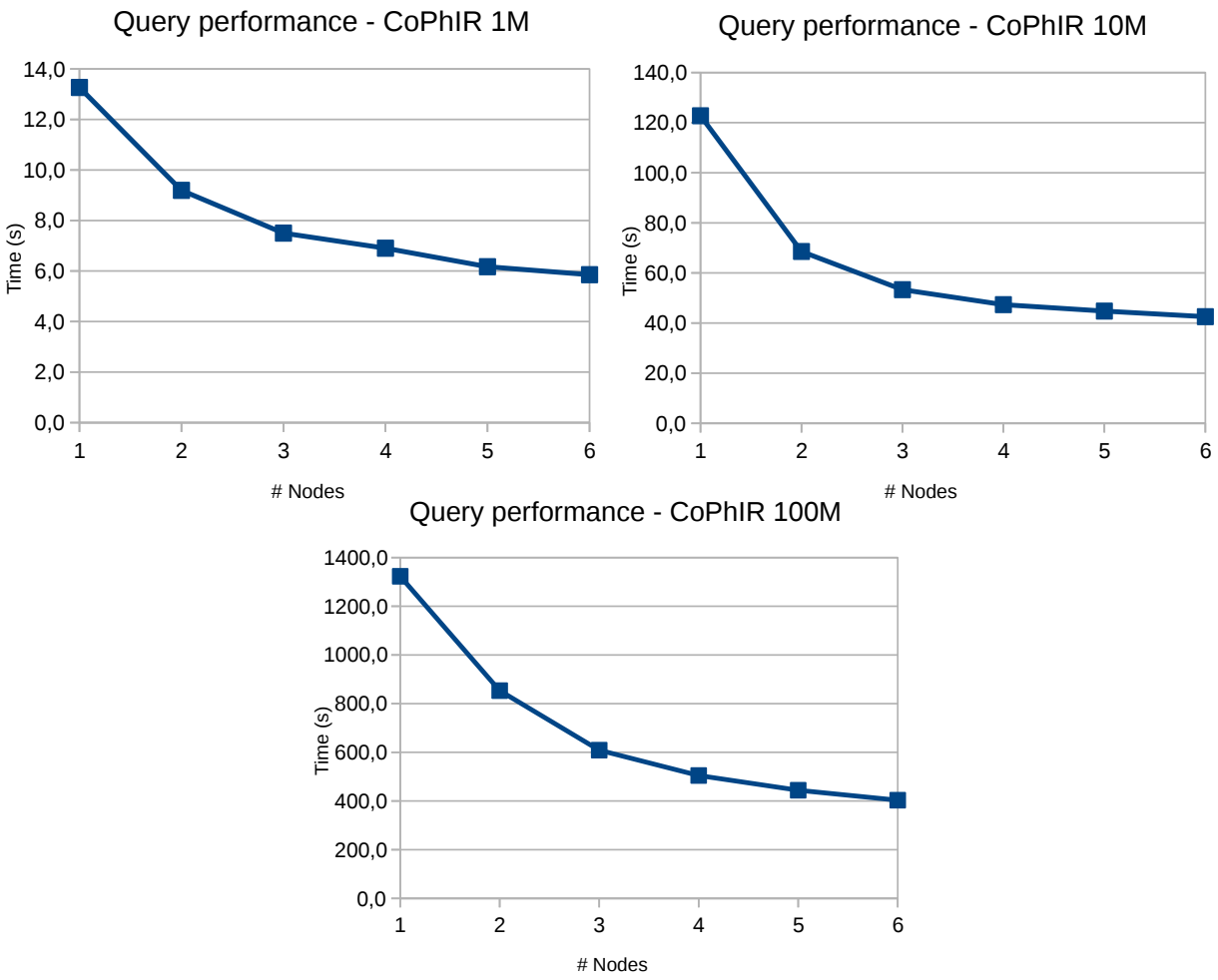
These curves indicate that queries in larger datasets can obtain greater improvements from using more processing nodes. When querying around 1 million elements, there was a performance improvement of two times when using six processing nodes when comparing to using only one.

With 10 million elements, this improvement almost reaches three times, and with 100 million elements it goes even beyond three times. We believe this occurs because the demands the whole dataset creates to a single node are bigger than the management overhead of using six nodes, when all of them are used for queries. This behavior is even more obvious in Figure 18, where we can see the calculated speedup increasing by higher margins when the dataset grows.

We also evaluated how our prototype performance scales with increased dataset size. Figure 19a shows extraction scales linearly with dataset size (notice the graph in log scale). We can see a smaller slope from 2 to 20 thousand images than from 20 to 200 thousand images. This is due the environment setup overhead being more apparent when the dataset is small. Similarly, Figure 19b shows how the execution of sequential queries presented a strictly linear pattern (also in log scale). Although the speedup achieved by increasing the number of nodes seems to be constant with increasing dataset size, the correlation is not perfect (the graph lines are not parallel) and we know the speedup increases for large datasets (see Figure 18).

Next we show how the usage of the plugin approach for searching is able to greatly increase our prototype performance. The MAM used in this experiment is the Slim-Tree [44], using an implementation with a disk page manager (which creates a local caching mechanism) and the Slim-Down optimization algorithm. We employed 64 KB as the page size, which is a general value (*i.e.*, not optimized for the specific application). In Figures 20 and 21 we compare the time for executing queries using both the sequential and plugin-based approaches. We took 50 random query centers, and executed 50 sequential queries. And executed 50 plugin-based queries with the same centers. The times on both pictures represent the average time of these executions.

We can notice how on the execution with the 1 million dataset, the plugin-based search is almost 24 times faster on average. At the same time, the improvement on the execution with 10 million objects is of almost 17 times. This difference of more than one order of magnitude in both cases is due the fact that the plugin-based search uses



(c) 100 million

Figure 17 – Performance of k-NN queries in CoPhIR varying number of nodes and dataset size

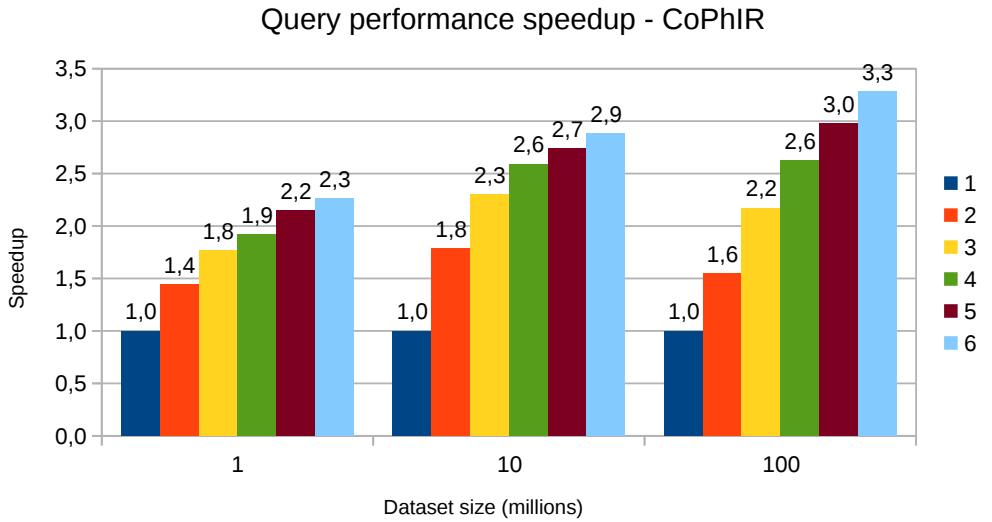
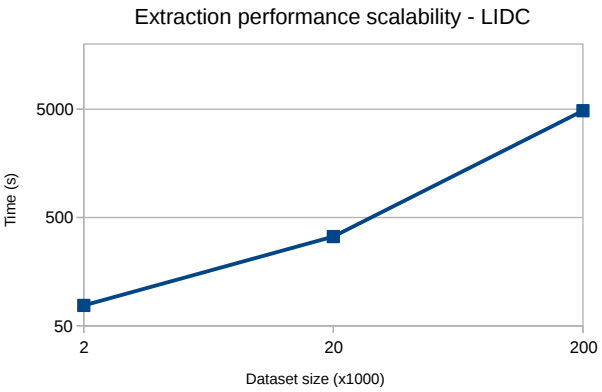


Figure 18 – Speedup in k-NN queries in CoPhIR for varying number of nodes and dataset size

(a) Feature Extraction in LIDC-IDRI



(b) Query Execution in CoPhIR

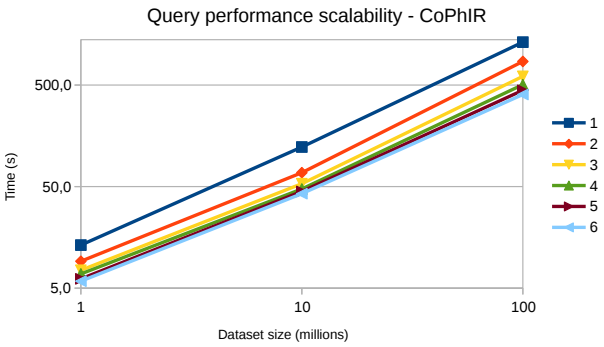


Figure 19 – Scalability in feature extraction and querying

the indexing capabilities of the underlying MAM (SlimTree) to prune the search space. Therefore, there is no need to inspect all elements as is the case of the sequential approach.

Obviously, what dictates how much improvement there will be is the MAM plugged in the architecture, but as we described in Section 3.3.2.2, one of our aims was to provide means for easily plugging in any kind of MAM, without changes on its own source code. Users wanting to use other kinds of MAM should only worry about implementing a thin layer of glue code that makes it possible for the architecture to invoke new processes of the new index data structure.

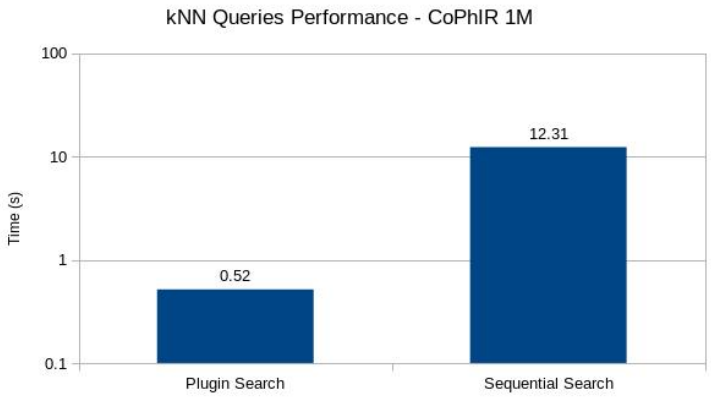


Figure 20 – Plugin Search in dataset of 1 million objects

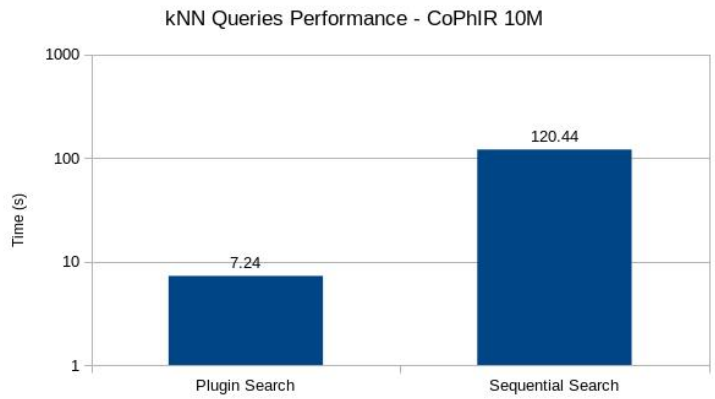


Figure 21 – Plugin Search in dataset of 10 million objects

These results confirm our proposal possesses expressiveness to represent the Similarity Space Definition and scales well to big data, relinquishing end-users from programming and administering computational infrastructure. The test cases we showed in the experimental evaluation depict a range of different operations that are common and that we judge necessary in the context of similarity searching in complex data sets, in general, and image data set, in our specific case.

5 CONCLUSION

This work was motivated by the increasing rate of production of complex data, such as images, videos and other multimedia content, which requires appropriate data management techniques to be employed. It is through the concept of similarity that complex data can be efficiently stored and retrieved, and this requires that a similarity space be defined in the first place.

Although many recent works address the concern of properly searching by similarity in large-scale data sets, they often rely on a fixed concept of similarity. On the other hand, approaches related to building similarity spaces by experimenting either don't scale or are limited to proprietary software and specialized hardware. We consider that there is still room for improvement in the operations that actually define that concept of similarity, mainly by exposing this process to end-users. Furthermore, several methods designed to increase similarity queries performance can benefit these same users by reducing time to complete retrieval tasks.

The main contributions of this work are as follows.

- The creation an architecture for defining similarity spaces for large image datasets by using distributed computation techniques. The architecture is orchestrated through scientific workflows, allowing that end-users coordinate computational tasks without specific programming the system, but still having access to many techniques that are commonly used in image retrieval tasks. Through this architecture, it is possible that one defines pipelines for several Content-Based Image Retrieval tasks.
- the creation of an application programming interface that enables that data structures and metric access methods never conceived to work in distributed settings be used as plugins in the same architecture. Therefore, not only we could achieve scalability in the definition of similarity spaces, but also in retrieval tasks that are unfeasible to be done without techniques specifically created to boost query performance.

As future work, we would like to experiment with other types of access methods and evaluate if their integration in our architecture could result in improved retrieval performance. We also find that better support for data updates should be part of future proposals. We are also interested in extending the architecture with the capability of searching multimodal data by similarity, which can allow that the same data be queried using multiple adjacent criteria. And finally, we would also like evaluate how users would

experience our initial proposal and, based on their feedback, study ways of approaching a better usability.

REFERENCES

[1] LUX, M. Content based image retrieval with lire. In: *Proceedings of the 19th ACM International Conference on Multimedia*. New York, NY, USA: ACM, 2011. (MM '11), p. 735–738. ISBN 978-1-4503-0616-4. Disponível em: <http://doi.acm.org/10.1145/2072298.2072432>.

[2] DATTA, R. et al. Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Computing Surveys*, ACM, v. 40, n. 2, p. 1–60, apr 2008. ISSN 03600300. Disponível em: <http://dl.acm.org/citation.cfm?id=1348246.1348248>.

[3] KUMAR, A. et al. Content-based medical image retrieval: A survey of applications to multidimensional and multimodality data. *Journal of Digital Imaging*, Springer US, v. 26, n. 6, p. 1025–1039, dec 2013. ISSN 08971889. Disponível em: <http://link.springer.com/10.1007/s10278-013-9619-2>.

[4] ZHANG, Z. et al. Scientific computing meets big data technology: An astronomy use case. In: *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*. IEEE, 2015. p. 918–927. ISBN 9781479999255. Disponível em: <http://ieeexplore.ieee.org/document/7363840/>.

[5] IQBAL, K.; ODETAYO, M. O.; JAMES, A. Content-based image retrieval approach for biometric security using colour, texture and shape features controlled by fuzzy heuristics. *Journal of Computer and System Sciences*, v. 78, n. 4, p. 1258–1277, jul 2012. ISSN 00220000. Disponível em: <http://www.sciencedirect.com/science/article/pii/S002200001100119X>.

[6] GAUKER, C. A critique of the similarity space theory of concepts. *Mind & Language*, Wiley Online Library, v. 22, n. 4, p. 317–345, 2007.

[7] NOVAK, D.; ZEZULA, P. M-Chord. In: *Proceedings of the 1st international conference on Scalable information systems - InfoScale '06*. New York, New York, USA: ACM Press, 2006. p. 19–es. ISBN 1595934286. Disponível em: <http://portal.acm.org/citation.cfm?doid=1146847.1146866>.

[8] GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 3. ed. [S.l.]: Addison-Wesley, 2007. ISBN 0-201-50803-6.

[9] Burak Akgül, C. et al. Content-based image retrieval in radiology: Current status and future directions. *Journal of Digital Imaging*, v. 24, n. 2, p. 208–222, 2011. ISSN 08971889.

[10] DESERNO, T. M. *Biomedical Image Processing*. [s.n.], 2008. ISBN 9783642158155. Disponível em: <http://medcontent.metapress.com/index/A65RM03P4874243N.pdf>.

[11] DESERNO, T. M.; ANTANI, S.; LONG, R. Ontology of gaps in content-based image retrieval. *Journal of Digital Imaging*, v. 22, n. 2, p. 202–215, 2009. ISSN 08971889.

[12] TRAINA, A. J. M. et al. *Feature Extraction and Selection for Decision Making*. 2011. 197–223 p. Disponível em: <http://www.springerlink.com/index/10.1007/978-3-642-15816-2>.

[13] COMANICIU, D.; MEER, P.; FORAN, D. J. Image-guided decision support system for pathology. *Machine Vision and Applications*, v. 11, n. 4, p. 213–224, dec 1999. ISSN 0932-8092. Disponível em: <http://link.springer.com/10.1007/s001380050104>.

[14] ALTO, H. Content-based retrieval and analysis of mammographic masses. *Journal of Electronic Imaging*, 2005. Disponível em: <http://electronicimaging.spiedigitallibrary.org/article.aspx?articleid=1098528>.

[15] DOYLE, S.; HWANG, M.; NAIK, S. Using manifold learning for content-based image retrieval of prostate histopathology. *MICCAI 2007 Workshop ...*, 2007.

[16] KWAK, D. Content-Based Ultrasound Image Retrieval Using a Coarse to Fine Approach. *Annals of the New ...*, 2002. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1111/j.1749-6632.2002.tb04898.x/full>.

[17] HARALICK, R. M.; SHANMUGAM, K.; DINSTEIN, I. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 3, p. 610–621, 1973.

[18] KHOTANZAD, A.; HONG, Y. H. Invariant Image Recognition by Zernike Moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, v. 12, n. 5, p. 489–497, 1990.

[19] GULD, M. O. et al. Combining global features for content-based retrieval of medical images. *CEUR Workshop Proceedings*, v. 1171, 2005. ISSN 16130073. Disponível em: <http://link.springer.com/chapter/10.1007/11878773_77>.

[20] ZEZULA, P. et al. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. [S.l.]: Springer, 2006. v. 32. Hardcover. (Advances in Database Systems, v. 32). ISBN 0387291466.

[21] WILSON, D. R.; MARTINEZ, T. R. Improved heterogeneous distance functions. *Journal of artificial intelligence research*, v. 6, p. 1–34, 1997.

[22] GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. *JMLR*, JMLR.org, p. 1157–1182, 2003. ISSN 1532-4435.

[23] BARIONI, M. C. N. et al. Advanced database query systems: Techniques, applications and technologies. In: YAN, L.; MA, Z. (Ed.). Hershey, Pennsylvania, USA: IGI Global, 2011. cap. Querying Multimedia Data by Similarity in Relational DBMS.

[24] LIU, J. et al. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, v. 13, n. 4, p. 457–493, 2015. ISSN 1572-9184. Disponível em: <http://dx.doi.org/10.1007/s10723-015-9329-8>.

[25] VALENTE, F. et al. A dataflow-based approach to the design and distribution of medical image analytics. In: *8th Iberian Grid Infrastructure Conference Proceedings*. [S.l.: s.n.], 2014. p. 201.

[26] SRIDHARAN, R. et al. *Visualization and Analysis of Large Medical Image Collections Using Pipelines*. Tese (Doutorado) — Massachusetts Institute of Technology, 2015.

[27] HANBURY, A. et al. Cloud-based evaluation framework for big data. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Berlin, Heidelberg, 2013. v. 7858 LNCS, p. 104–114. ISBN 9783642380815. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-642-38082-2_9>.

[28] MERA, D.; BATKO, M.; ZEZULA, P. Speeding up the multimedia feature extraction: a comparative study on the big data approach. *Multimedia Tools and Applications*, Springer US, p. 1–21, mar 2016. ISSN 15737721. Disponível em: <http://link.springer.com/10.1007/s11042-016-3415-1>.

[29] ZHAO, W.; MA, H.; HE, Q. Parallel K-Means Clustering Based on MapReduce. In: . Springer, Berlin, Heidelberg, 2009. p. 674–679. Disponível em: <http://link.springer.com/10.1007/978-3-642-10665-1_71>.

[30] MOISE, D. et al. Indexing and searching 100M images with map-reduce. *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval - ICMR '13*, ACM Press, New York, New York, USA, p. 17, 2013. Disponível em: <http://dl.acm.org/citation.cfm?doid=2461466.2461470>.

[31] SONG, G. et al. K Nearest Neighbour Joins for Big Data on MapReduce: A Theoretical and Experimental Analysis. *IEEE Transactions on Knowledge and Data Engineering*, v. 28, n. 9, p. 2376–2392, sep 2016. ISSN 1041-4347. Disponível em: <http://ieeexplore.ieee.org/document/7464884/>.

[32] MAILLO, J. et al. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems*, 2016. ISSN 09507051.

[33] The Apache Software Foundation. *Apache™ Hadoop®*. 2008. <http://hadoop.apache.org/>. Acessado: 2017-08-20.

[34] ZAHARIA, M. et al. Spark: Cluster computing with working sets. *HotCloud*, v. 10, n. 10-10, p. 95, 2010.

[35] RANJAN, R. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, IEEE, v. 1, n. 1, p. 78–83, 2014.

[36] CARBONE, P. et al. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, IEEE Computer Society, v. 36, n. 4, 2015.

[37] MANJUNATH, B. S.; SALEMBIER, P.; SIKORA, T. *Introduction to MPEG-7: multimedia content description interface*. [S.l.]: John Wiley & Sons, 2002. v. 1.

[38] BATKO, M.; NOVAK, D.; ZEZULA, P. Messif: Metric similarity search implementation framework. *Digital Libraries: Research and Development*, Springer, p. 1–10, 2007.

[39] NOVAK, D. Multi-modal Similarity Retrieval with Distributed Key-value Store. *Mobile Networks & Applications*, Springer US, v. 20, n. 4, p. 521–532, aug 2015. ISSN 1383-469X. Disponível em: <http://link.springer.com/10.1007/s11036-014-0561-4>.

[40] YIANILOS, P. N. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In: *Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)*. Austin, TX: [s.n.], 1993. p. 311–321.

[41] BOZKAYA, T.; ÖZSOYOGLU, Z. M. Distance-Based Indexing for High-Dimensional Metric Spaces. In: *ACM SIGMOD International Conference on Management of Data*. Tucson, AZ: ACM Press, 1997. p. 357–368.

[42] BAEZA-YATES, R. A. et al. Proximity Matching Using Fixed-Queries Trees. In: *Combinatorial Pattern Matching (CPM)*. Asilomar, CA: Springer Verlag, 1994. (Lecture Notes in Computer Science, v. 807), p. 198–212.

[43] CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: *International Conference on Very Large Databases (VLDB)*. Athens, Greece: Morgan Kaufmann, 1997. p. 426–435.

[44] Traina Jr., C. et al. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In: ZANIOLO, C. et al. (Ed.). *International Conference on Extending Database Technology (EDBT)*. Konstanz, Germany: Springer Verlag, 2000. (Lecture Notes in Computer Science, v. 1777), p. 51–65.

[45] HAR-PELED, S.; INDYK, P.; MOTWANI, R. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of Computing*, v. 8, n. 1, p. 321–350, 2012.

[46] SUNDARAM, N. et al. Streaming Similarity Search over one Billion Tweets using Parallel Locality-Sensitive Hashing. *PVLDB*, v. 6, n. 14, p. 1930–1941, 2013.

[47] ESULI, A. Use of permutation prefixes for efficient and scalable approximate similarity search. *Inf. Process. Manage.*, v. 48, n. 5, p. 889–902, 2012.

[48] NOVAK, D.; BATKO, M.; ZEZULA, P. Metric Index: An efficient and scalable solution for precise and approximate similarity search. *Information Systems*, v. 36, n. 4, p. 721–733, jun 2011. ISSN 03064379. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0306437910001109>.

[49] NOVAK, D.; BATKO, M.; ZEZULA, P. Large-scale similarity data management with distributed Metric Index. *Information Processing & Management*, v. 48, n. 5, p. 855–872, 2012. ISSN 03064573. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306457310001056>.

[50] JAGADISH, H. V. et al. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, v. 30, n. 1, p. 364–397, 2005. ISSN 0362-5915.

[51] OLIVEIRA, L. F. M.; KASTER, D. d. S. Defining similarity spaces for large-scale image retrieval through scientific workflows. In: *Proceedings of the 21st International Database Engineering & Applications Symposium*. New York, NY, USA: ACM, 2017. (IDEAS 2017), p. 57–65. ISBN 978-1-4503-5220-8. Disponível em: <http://doi.acm.org/10.1145/3105831.3105863>.

[52] THAMSEN, L. et al. Visually programming dataflows for distributed data analytics. In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016. p. 2276–2285. ISBN 978-1-4673-9005-7. Disponível em: <http://ieeexplore.ieee.org/document/7840860/>.

[53] BOLT, A.; LEONI, M. de; AALST, W. M. van der. Scientific workflows for process mining: building blocks, scenarios, and implementation. *International Journal on Software Tools for Technology Transfer*, Springer, v. 18, n. 6, p. 607–628, 2016.

[54] SCHNEIDER, C. A.; RASBAND, W. S.; ELICEIRI, K. W. Nih image to imagej: 25 years of image analysis. *Nature methods*, Nature Publishing Group, v. 9, n. 7, p. 671, 2012.

[55] GRAF, F. *JFeatureLib v1.6.3*. 2015. Disponível em: <https://doi.org/10.5281/zenodo.31793>.

[56] ZAYED, N.; ELNEMR, H. A. Statistical analysis of haralick texture features to discriminate lung abnormalities. *Journal of Biomedical Imaging*, Hindawi Publishing Corp., v. 2015, p. 12, 2015.

[57] CLARK, K. et al. The cancer imaging archive (tcia): maintaining and operating a public information repository. *Journal of digital imaging*, Springer, v. 26, n. 6, p. 1045–1057, 2013.

[58] ARMATO, S. G. et al. The lung image database consortium (lidc) and image database resource initiative (idri): a completed reference database of lung nodules on ct scans. *Medical physics*, Wiley Online Library, v. 38, n. 2, p. 915–931, 2011.

[59] BOLETTIERI, P. et al. CoPhIR: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009. Disponível em: <http://cophir.isti.cnr.it>.

PUBLICATIONS

Works published by the author during the Master's Degree:

Main publications.

1. Luis Fernando Milano Oliveira, Daniel dos Santos Kaster, **Defining Similarity Spaces for Large-Scale Image Retrieval Through Scientific Workflows**, Proceedings of the 21st International Database Engineering & Applications Symposium. ACM, 2017. NBR 6023 (Qualis CC, B2)

Complementary publications.

1. CAZZOLATO, Mirela T. ; SCABORA, L. C. ; COSTA, Alceu Ferraz ; NESSO JR, M. R. ; MILANO-OLIVEIRA, L. F. ; Kaster, D. ; TRAINA JR, Caetano ; TRAINA, A. J. M.. BREATH: Heat Maps Assisting the Detection of Abnormal Lung Regions in CT Scans. Em: 30th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2017), v. 1, p. 248-253, 2017 (Qualis CC, B1)