Efficient Processing of Analytical Queries Extended with Similarity Search Predicates over Images in Spark

Guilherme Muzzi da Rocha, Cristina Dutra de Aguiar Ciferri

University of São Paulo, Brazil
guilherme.muzzi.rocha@usp.br, cdac@icmc.usp.br

Abstract. An image data warehousing extends a conventional data warehousing to also manipulate images represented by feature vectors and attributes for similarity search. A challenge that arises is the efficient processing of analytical queries extended with a similarity search predicate. These queries have a high computational cost since they require the processing of costly star join operations and distance calculations in the same setting. We consider applications that manage huge volumes of data, where the use of parallel and distributed data processing frameworks is needed. In this article, we introduce two methods to efficiently solve this challenge in Spark. $BrOmnImg$ is based on the integration of the broadcast join and the Omni techniques for the processing of the star join operation and the distance calculations, respectively. $BrOmnImg^{CF}$ extends $BrOmnImg$ by using the conventional predicate to further reduce the number of distance calculations. Compared with the closest method available in the literature, $BrOmnImg$ reduced the time spent on query processing by up to about 65%. Compared with $BrOmnImg$, $BrOmnImg^{CF}$ improved the performance by up to about 54%.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Query processing; H.4.2 [**Information Systems Applications**]: Decision support

Keywords: Image data warehouse, analytical queries extended with a similarity search predicate, parallel and distributed processing, medical images, star join, distance calculations

1. INTRODUCTION

A data warehousing is of paramount importance to the decision-making process. It stores data from autonomous, distributed, and heterogeneous sources in the data warehouse. Data in the warehouse is integrated, subject-oriented, non-volatile, historical, and multidimensional [Kimball and Ross 2002]. Analytical queries, called OLAP (on-line analytical processing), are issued against the data warehouse to discover useful trends. In relational implementations of the warehouse, data are usually modeled through a star schema, where a central fact table is linked to satellite dimension tables. Therefore, the conventional predicate of an OLAP query requires the processing of the star join operation, i.e., performing joins between the fact table and each conventional dimension table involved in the query.

An image data warehousing extends a conventional data warehousing to also manipulate images [Teixeira et al. 2015]. The ETL (extract, transform, load) process is also responsible for extracting the intrinsic features of the images, which are represented by features vectors and attributes for similarity search. In the image data warehouse, these new types of data are stored as dimension tables or facts in the fact table. Further, OLAP queries are composed of a conventional predicate and a similarity search predicate. The similarity between two images can be evaluated by a distance function that becomes smaller as the images become more similar [Hjaltason and Samet 2003]. Therefore, the similarity search predicate requires the processing of distance calculations.

A new range of queries can be performed over the image data warehouse, enriching the decision-

making. For instance, a medical organization focused on analyzing images of exams can determine how many images of pulmonary cancer are similar to a given image, considering patients older than 40 years and that lived in São Paulo state between 2010 and 2013 [Teixeira et al. 2015]. In the agriculture field, agricultural measures related to soil, fertility and humidity over the years represent conventional data types and the image feature vectors extracted from images of the plantation represent extended data types [Nguyen et al. 2017]. In the decision-making, they can be used to investigate the quantity of images that are similar to a given soil image, considering specific values for the agricultural measures.

A core issue in an image data warehousing is to process OLAP queries extended with a similarity search predicate. In addition to execute the expensive star join operation, these queries also require the computation of costly distance calculations. Furthermore, image data warehouses are very voluminous, as they store conventional and image data. They also may be fed from applications in big data, corroborating the fact that their volume may be orders of magnitude larger than megabytes [Kuo et al. 2014].

Data warehousing applications that exceed the megabyte order and involve costly operations usually require storage and processing capacity to guarantee scalability [Cuzzocrea 2016]. They can benefit from using the MapReduce programming model [Dean and Ghemawat 2008] and data processing frameworks like the Apache Hadoop[1] and the Apache Spark[2] [Zaharia et al. 2010]. These frameworks enable data sharing and robust decision-making [Sebaa et al. 2018]. In the literature, the use of these frameworks has become an attractive alternative to individually minimize the cost of the star join operation over conventional data warehouses and the cost of distance calculations in applications that manage complex data. However, to the best of our knowledge, there are no approaches that investigate these aspects in the same setting.

In this article, we fill this gap. We propose the $BrOmnImg$ and the $BrOmnImg^{CF}$ methods to efficiently process analytical queries extended with a similarity search predicate in Spark. $BrOmnImg$ integrates the broadcast join [Brito et al. 2016] and the Omni [Traina-Jr et al. 2007] techniques. Broadcast join is used to load small tables on each node of the cluster and to compute the joins in parallel. Omni is used to decrease the number of distance calculations in image similarity searches. $BrOmnImg^{CF}$ empowers $BrOmnImg$ to deal with applications that restrict to one-to-one the relationship between the fact table and the table that stores the feature vectors of the images.

The advantages of $BrOmnImg$ and $BrOmnImg^{CF}$ were investigated through performance tests considering a medical image data warehouse. The experiments exploited different properties that affect the general behavior of queries, i.e., the selectivity of the similarity search predicate and of the conventional predicate. We compared $BrOmnImg$ to the closest method available in the literature and also compared $BrOmnImg^{CF}$ with $BrOmnImg$. The results demonstrated that the proposed methods are very competitive solutions to process analytical queries extended with a similarity search predicate in image data warehousing applications in Spark. Compared with the closest method available in the literature, $BrOmnImg$ reduced the time spent on query processing by up to 64,47%. Compared with $BrOmnImg$, $BrOmnImg^{CF}$ improved the performance by up to 54.21%.

A preliminary version of this article is described in [Rocha and Ciferri 2019]. Here, we provide a detailed description of the background. We also detail the systematic review. Furthermore, we introduce the algorithm of $BrOmnImg$. Moreover, we propose the novel $BrOmnImg^{CF}$. Finally, we describe performance tests that include a different data volume and new values for the selectivity of the conventional predicate, as well as compare $BrOmnImg$ with $BrOmnImg^{CF}$.

The paper is organized as follows. Section 2 details the theoretical foundation, Section 3 describes the systematic review, Section 4 proposes $BrOmnImg$ and $BrOmnImg^{CF}$, Section 5 describes the performance evaluation, and Section 6 concludes the article.

[1]https://hadoop.apache.org/
[2]https://spark.apache.org/

2. THEORETICAL FOUNDATION

In this section, we summarize underlying concepts used in our proposal. We describe the main issues related to similarity search over images and the Omni technique in Section 2.1. We introduce details related to a image data warehousing in Section 2.2. Sections 2.3 and 2.4 are aimed to describe parallel and distributing computing and the broadcast join technique, respectively.

2.1 Similarity Search and The Omni Technique

Complex data, such as image data considered in this work, can not be sorted and searched using ordinary relational operators (e.g., $<$, $\leq$, $>$, $\geq$). They should be compared through a distance function that calculates the dissimilarity between two complex elements based on their feature vectors [Hjaltason and Samet 2003]. Examples of widely used distance functions are the Manhattan and the Euclidean distances, known as L_1 and L_2, respectively. A feature vector describes a given characteristic of interest. For instance, feature vectors generated by image extractors, such as Color Histograms [Gonzalez and Woods 2006] and Haralick descriptors [Haralick 1979], contain the numeric representations of these images according to the attributes of color and texture, respectively.

One of the most useful types of similarity query is the range query. It is defined as follows. Given a dataset S of elements (i.e., the feature vectors of the images), a distance function d, and a query radius r_q, this query retrieves every element $s_i \in S$ that satisfies the condition $d(s_i, s_q) \leq r_q$. To this end, the range query calculates the costly distance function several times, one for each pair (s_q, s_i). The Omni technique drastically reduces the number of distance calculations [Traina-Jr et al. 2007]. It is based on strategically positioned elements $f_j \in S$, called *foci*, and on previously storing the distances between each s_i and each f_j. It is composed of two steps. In the filtering step, each f_j is used to generate a ring in the metric space that surrounds s_q, considering r_q and the distance between f_j and s_q. The intersection of the rings produces a region with candidate elements to answer the query. In the refinement step, distances are calculated only for the candidate elements to eliminate false positives. The number of *foci*, determined by the Hull-Foci algorithm, is given by the intrinsic dimensionality of the dataset.

2.2 Image Data Warehousing

In addition to conventional dimension tables and fact tables, an image data warehouse also contains tables that store the feature vectors of images and attributes for similarity search [Teixeira et al. 2015]. Figure 1 depicts a schema of a medical image data warehouse that is used throughout the paper. It employs different colors and line styles to identify the different types of tables. The conventional dimension tables, *Patient* and *ExamDescription*, are represented using the blue color and long dash dot dot lines. The feature vectors of all images generated by all image extractors are stored in the $FeatureVector$ table, which are colored in red and are designed using thick lines. In this running example, there are two feature vectors for each image. The attributes for similarity search, i.e., the distances between each image and each *foci* determined by the Omni technique, are stored in the *PerceptualLayer* tables $Color\ Histogram$ and $Haralick\ Variance$. The green color and dash lines are used to identify these tables. Finally, the fact table *Exam*, colored in yellow and designed using thin lines, stores the quantity of exams by patient, by exam description, and by exam image represented by the attributes of color and texture. In detail, a perceptual layer is an abstraction that represents images through their feature vectors, which are generated according to a specific feature descriptor (e.g., Color Histograms and Haralick Variance), and their similarity search data, which are generated according to a specific metric space.

The image data warehouse supports the processing of analytical queries extended with a similarity search predicate. These queries may have two predicates. The conventional predicate is based on selection conditions. Each selection condition is defined over an attribute stored in a conventional

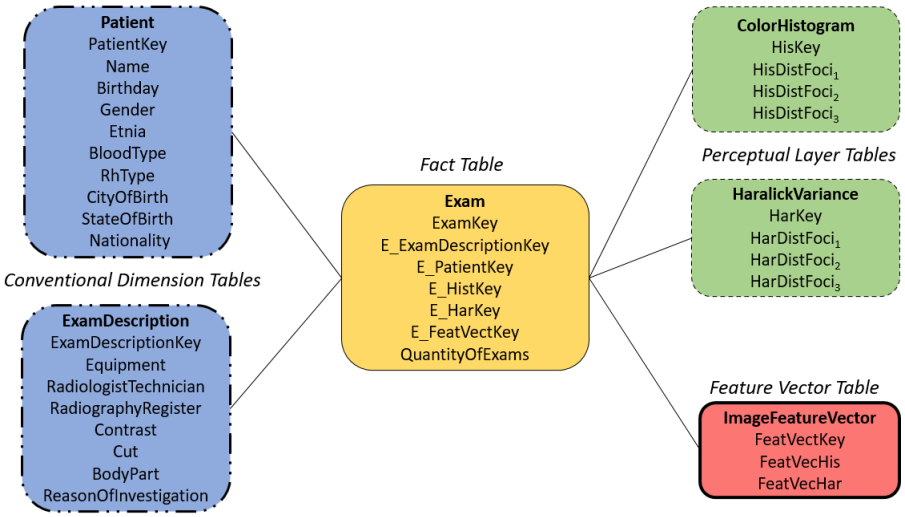

Fig. 1. Schema of the medical image data warehouse used throughout the paper.

dimension table. The similarity search predicate is composed of a similarity operation and one or more perceptual layers. For instance, consider the query named as $Q_{example}$: "List the number of similar images to a given image, for female patients diagnosed with breast cancer that have Rh factor blood positive". The selection conditions of $Q_{example}$ are: (i) $Gender$ = 'female' and $RhType$ = 'positive', defined over the table $Patient$; and (ii) $BodyPart$ = 'breast' and $ReasonInvestigation$ = 'cancer', defined over the table $ExamDescription$. The similarity operation is *range query* and is performed using the table $FeatureVector$. Finally, because the images are represented considering the attributes of color and texture, $Q_{example}$ uses the tables $ColorHistograms$ and $HaralickVariance$.

In [Teixeira et al. 2015], it was conducted a study that investigated the use of a image data warehousing from the point of view of medical specialists. Two high level queries like $Q_{example}$ were defined to support the specialists' opinions. According to them, it is feasible to use a image data warehousing and the queries have great potential for improving the medical decision-making.

2.3 Parallel and Distributed Computing

The Hadoop Distributed File System (HDFS) [Shvachko et al. 2010] has been currently used as a support for parallel and distributed computing environments. In the HDFS, the data file is divided into blocks, which are distributed and replicated on the nodes of the cluster. There are a *namenode* and several *datanodes*. The *namenode* stores metadata related to the file, such as the nodes where each block is stored, while the *datanodes* store the distributed and replicated blocks. There may exist a secondary *namenode* used as backup. To manipulate a given file, it is necessary to access the *namenode* to identify the nodes where each block of the file is stored.

Hadoop is a parallel and distributed data processing framework based on the MapReduce programming model [Dean and Ghemawat 2008]. It is able to process big datasets distributed across a cluster of machines through the execution of map and reduce functions. Map functions process input data and transform them into intermediate key-value outputs, while reduce functions process these intermediate key-values by integrating all values related to a given key into a single output. In this model, all intermediate data are written to disk in the HDFS [Shvachko et al. 2010].

On the other hand, Spark is a in-memory parallel and distributed data processing framework based on the concept of Resilient Distributed Datasets (RDDs) [Li et al. 2017]. Further, all operations on the RRDs are first mapped into a directed acyclic graph and then reorganized into sets of smaller

tasks according to their mutual dependencies. In this model, intermediate results are stored in the main memory and data are written into HDFS only when necessary, considerably reducing the I/O cost [Shvachko et al. 2010]. Data manipulation in Spark is performed with predefined operations defined over the RDDs. Operations used in this work are: (i) *filter*: returns a new RDD containing only the elements that satisfy a conventional predicate; (ii) *collect*: sends all data from the dataset to a driver process; (iii) *mapToPair*: processes the data and generates intermediate key-value pairs; and (iv) *reduceByKey*: merges the values for each key using a reduce function.

2.4 The Broadcast Join Technique

The broadcast join technique called SBJ [Brito et al. 2016] processes star joins over conventional data warehouses in Spark as follows (Figure 2). Consider that the conventional dimension tables and the fact table are distributed in the nodes of the cluster. The selection condition is performed on each block i that contains the dimension table $Conventional_i$, generating a conventional dimension table $Conv_{123}$ that contains the filtered data. SBJ assumes that this table is small enough to fit in the main memory. Then, it broadcasts $Conv_{123}$ to all nodes of the cluster that contain blocks of the fact table. Finally, SJB computes in parallel the joins between $Conv_{123}$ and the blocks of the fact table on each node.

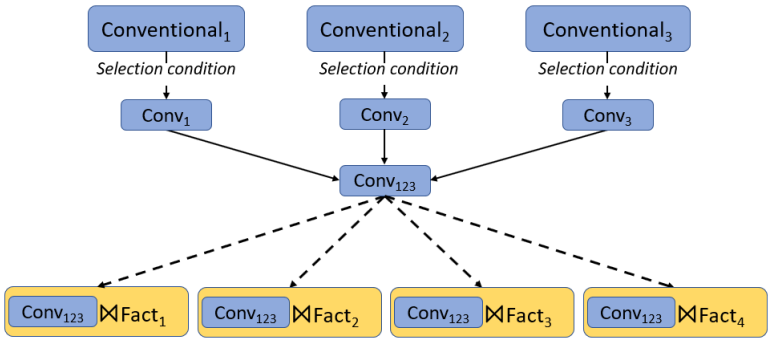

Fig. 2. General view of the broadcast join technique.

3. SYSTEMATIC REVIEW

The goal of a systematic review is to identify gaps in the literature that should be explored and to discover approaches that have similar characteristics to the proposed methods [Kitchenham and Charters 2007]. Furthermore, our work was developed considering the context of medical images, due to the importance of the analytical decision-making over these images and their impact on society. For instance, it is possible to generate knowledge that can be used to identify trends in healthcare, prevent diseases, combat social and health inequalities, and provide new ideas about science [Kuo et al. 2014]. It is also possible to make vital decisions against disease outbreaks and treatment effectiveness [Teixeira et al. 2015]. Another example is the analysis of patient profiles to identify individuals who would benefit from preventative care and lifestyle changes [Raghupathi and Raghupathi 2014].

To comply with the goal of a systematic review and to consider the medical context, we defined the following search questions: (Q_1) How to perform the star join operation in Hadoop; (Q_2) How to perform image similarity search operations in Hadoop; (Q_3) How to design a medical data warehouse in Hadoop; and (Q_4) How to perform the star join and the similarity search operations over a data warehouse in Hadoop. The term "Hadoop" was used because there are more related work in the literature that use this term instead of "parallel and distributed processing".

Based on the search questions, we defined the keywords depicted in Figure 3 to compose the search strings. We submitted the strings to the following sources: IEEEXplore Digital Library[3], Springer[4], ACM Digital Library[5], and Elsevier[6]. We did not consider the Digital Bibliography & Library Project[7] (DBLP) because most of its publications can be obtained from these other sources [Batista et al. 2018]. As inclusion criterion, we considered the articles that answer at least one search question (i.e., Q_1, Q_2, Q_3, or Q_4). As exclusion criteria, we defined: (i) articles that do not answer any search question; (ii) articles that are not complete or are not fully available; (iii) articles written in other languages than Portuguese or English; and (iv) articles published before 2014.

Figure 3 details the systematic review process. The search returned 148 articles. The initial selection was performed by reading the title and the abstract of the articles. According to the exclusion criteria, 128 articles were discarded. The final selection was performed by reading the full article, leading to the exclusion of 6 more articles. The remaining articles were grouped based on the defined search questions as follows: star join in Hadoop (Section 3.1), image similarity search in Hadoop (Section 3.2), medical data warehousing in Hadoop (Section 3.3), and star join and similarity operations over data warehouses in Hadoop (Section 3.4). Figure 3 also lists the related articles for each group, which are ordered according to the year of publication.

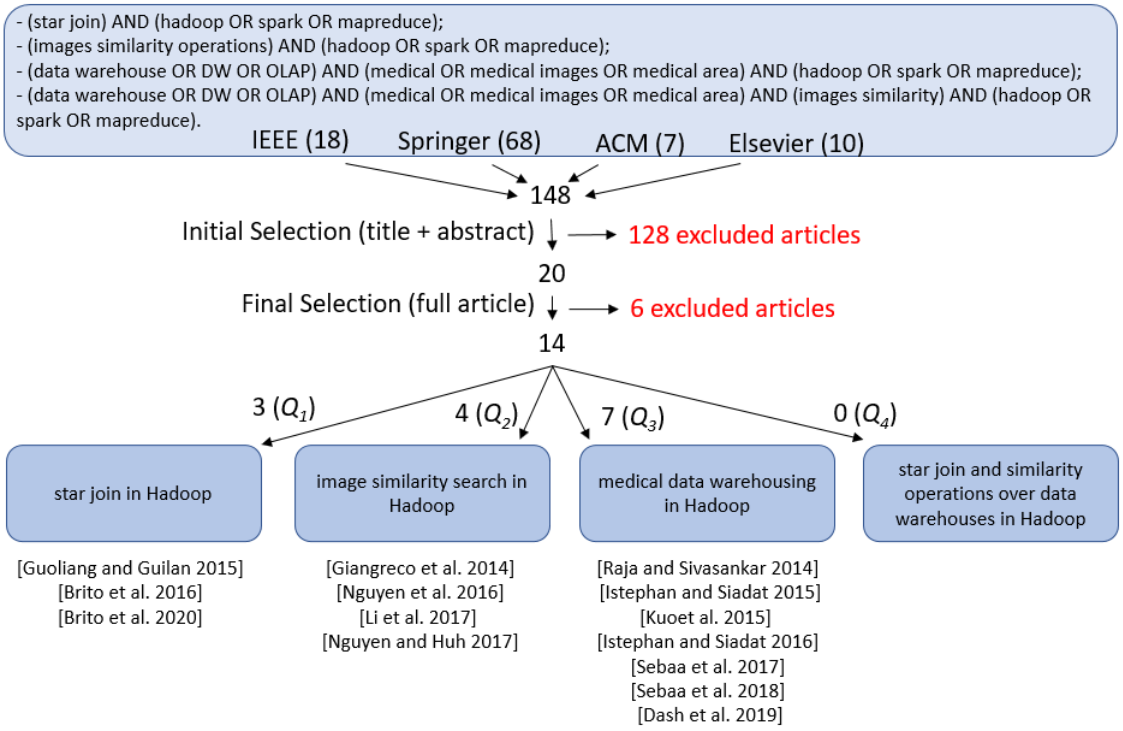

Fig. 3. Systematic review execution.

[3]IEEEXplore Digital Library: (https://ieeexplore.ieee.org)
[4]Springer: (https://www.springer.com/ComputerScience)
[5]ACM Digital Library: (https://dl.acm.org)
[6]Elsevier: (https://www.elsevier.com/physical-sciences/computer-science)
[7]DBLP: (https://dblp.uni-trier.de/)

3.1 Star Join in Hadoop

Articles classified in this group are aimed to propose efficient techniques to process the star join operation considering new technologies. The massive parallelism and high scalability GBFSJ (GPUs Bloom Filter Star Join) algorithm in proposed in [Guoliang and Guilan 2015]. It considers that the fact table and the dimension tables are column storage and that every column is processed separately. The star join is composed of several parallel hash join procedures executed on GPUs (Graphics Processing Units) with the help of Bloom filters [Tarkoma et al. 2012]. But, this related work differ from our work on its purpose since it is focused on GPUs.

The processing of star joins in Hadoop is exhaustively investigated in [Brito et al. 2016; 2020]. In these articles, several algorithms are proposed and compared with a wide range of related work available in the literature. In [Brito et al. 2016], the proposed algorithms are characterized by performing full scan regardless of the query selectivity. In [Brito et al. 2020], the proposed algorithms are based on the use of a distributed Bitmap Join Index. We use in our work the SBJ algorithm [Brito et al. 2016], which is described in Section 2.4.

3.2 Image Similarity Search in Hadoop

A system that is able to store and retrieve multimedia objects is introduced in [Giangreco et al. 2014]. It provides functionalities to Boolean retrieval and similarity search in MapReduce. Despite the advantages introduced in this related work due to the use of the parallel and distributed computing, it does not focus on reducing the number of distance calculations. In our work, we use the Omni technique to comply with this goal.

The approach proposed in [Li et al. 2017] optimize image similarity operations in Spark by applying hash functions that guarantee a high probability that similar objects will collide. Therefore, there is a reduction in the number of objects to be analyzed. In [Nguyen et al. 2016; Nguyen and Huh 2017], the VP-tree method [Fu et al. 2000] is used to reduce the number of distance calculations in MapReduce. This method is also associated with a strategy that is used to store distances previously calculated in cache [Nguyen et al. 2016]. However, it is difficult to efficiently define the hash functions. Also, the VP-tree is a in-memory method that is invariably outperformed by other methods [Carélo et al. 2011]. We use in our work the Omni technique [Traina-Jr et al. 2007]. It enables building similarity search operations on top of existing structures, providing prunability and therefore significantly improving their performance. The Omni technique is described in Section 2.1.

Although the articles in this group focus on parallel and distributed processing to optimize image similarity search, they are not based on the use of a image data warehouse. In detail, they are not aimed to optimize star join operations, which are demanding operations commonly present in OLAP queries over data warehouses. Furthermore, these articles do not consider the organization of the image data as fact and dimension tables. These issues are addressed by our proposal.

3.3 Medical Data Warehousing in Hadoop

Several articles in the literature investigate how the use of big data technology improves the performance of medical data warehousing [Raja and Sivasankar 2014; Istephan and Siadat 2015; 2016; Kuo et al. 2015; Sebaa et al. 2017; Sebaa et al. 2018; Dash et al. 2019]. The data warehouse defined in [Istephan and Siadat 2015; 2016] considers conventional data as structured data and medical images as unstructured data. These images are used to perform content based image retrieval in MapReduce. No details are provided regarding data storage. Dash et al. (2019) propose an architecture based on HDFS and Spark, as well as list several parallel and distributed-based tools aimed to perform medical analyses.

In [Raja and Sivasankar 2014], data from medical sources are transferred from a relational database

to the Apache HBase[8] using the Apache Sqoop[9]. The article compares the time spent: (i) to populate the relational database and HBase; and (ii) to process queries in MapReduce and in the Apache Hive[10]. In [Kuo et al. 2015], data from a medical application are stored in HBase and on-line transaction processing (OLTP) queries are performed against these data by using the Apache Phoenix[11]. The strategies described in [Sebaa et al. 2017; Sebaa et al. 2018] store data in Hive and query these data by using HiveQL. Furthermore, queries can also be carried out considering data stored in HBase.

Despite the fact that the articles described in this section highlight the importance of using parallel and distributed computing in medical data warehousing, they do not focus on analytical queries extended with a similarity search predicate. Therefore, they do not propose optimizations in the processing of these queries, differing from the objective of our work.

3.4 Star Join and Similarity Operations over Data Warehouses in Hadoop

No related work was classified in this last group. Thus, to the best of our knowledge, there are not approaches in the literature that investigate the processing of star join and distance operations in the same setting, considering image data warehousing and Spark. Our work fill this gap.

4. THE PROPOSED *BROMNIMG* AND *BROMNIMG*^{CF} METHODS

In this section, we propose the *BrOmnImg* and *BrOmnImg*^{CF} methods to efficiently process, in Spark, analytical queries extended with a similarity search predicate over a image data warehouse. We introduce these methods in Sections 4.1 and 4.2, respectively. We use as a basis the image data warehouse depicted in Figure 1. In Figures 4 and 5, we also employ the same colors and line styles as those used in Figure 1 when providing the general view of the methods: blue and long dash dot dot lines for the conventional dimension tables, red and thick lines for the feature vector table, green and dash lines for the perceptual layer tables, and yellow and thin lines for the fact table.

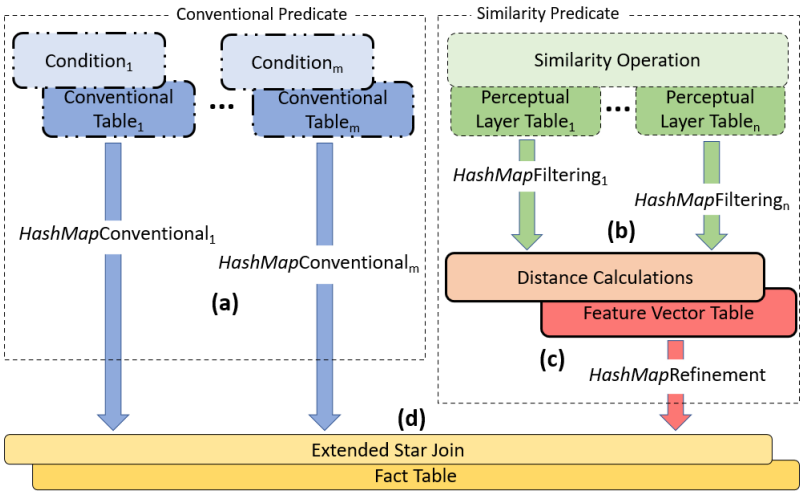

Fig. 4. General view of the proposed *BrOmnImg* method.

[8]https://hbase.apache.org/
[9]https://sqoop.apache.org/
[10]https://hive.apache.org/
[11]https://phoenix.apache.org/

4.1 *BrOmnImg*

Consider the conventional predicate. For each *Conventional* dimension table i ($1 \leq i \leq m$), *BrOmnImg* executes the conditions selections that apply on the attributes of this table and produces as a result a structure $HashMapConventional_i$ (Figure 4a). Consider now the similarity search predicate. Then, in the filtering step, for each *Perceptual Layer* table j ($1 \leq j \leq n$), *BrOmnImg* filters its data using the Omni technique and generates a structure $HashMapFiltering_j$ to store the candidate elements (Figure 4b). In the refinement step, the method analyzes all candidate elements to eliminate false positives. The *Feature Vector* table is accessed and distances are calculated to determine the elements to be further considered, which are stored in the structure $HashMapRefinement$ (Figure 4c). Finally, *BrOmnImg* broadcasts $HashMapRefinement$ and each $HashMapConventional_i$ to all nodes of the cluster and performs the extended star join over the *Fact* table in parallel (Figure 4d). Algorithm 1 details the algorithm of *BrOmnImg*.

Algorithm 1: *BrOmnImg*

Input : $Conventional_1$, ..., $Conventional_m$, $Perceptual\ Layer_1$, ..., $Perceptual\ Layer_n$, $Feature\ Vector$, $Fact$, s_q, r_q, Q
Output: Result of Q

1 **for** *each Conventional Table i between 1 and m* **do**
2 $\quad RDD_{Conventional_i} = Conventional_i$
3 $\quad RDD_{Conventional_i}.filter(Condition_{i_1}, ..., Condition_{i_p})$
4 $\quad RDD_{Conventional_i}.mapToPair(Conventional_iKey, null)$
5 $\quad HashMapConventional_i = broadcast(RDD_{Conventional_i}.collect()$
6 **end**
7 **for** *each Perceptual Layer Table j between 1 and n* **do**
8 $\quad RDD_{PerceptualLayer_j} = Perceptual\ Layer_j$
9 $\quad RDD_{PerceptualLayer_j}.filter(filteringStep(s_q, r_q, DistFoci_{j_1}, ..., DistFoci_{j_k}))$
10 $\quad RDD_{PerceptualLayer_j}.mapToPair(PerceptualLayer_jKey, null)$
11 $\quad HashMapFiltering_j = broadcast(RDD_{PerceptualLayer_j}.collect())$
12 **end**
13 $RDD_{FeatVect} = Feature\ Vector$
14 $RDD_{FeatVect}.filter(if(HashMapFiltering_1.hasKey(FeatVectKey)\ AND...\ AND\ HashMapFiltering_n.hasKey(FeatVectKey))\ then\ return\ (refinementStep(FeatVectPerceptualLayer_1, s_q, r_q)\ AND\ ...\ AND\ refinementStep(FeatVectPerceptualLayer_n, s_q, r_q))$
15 $RDD_{FeatVect}.mapToPair(FeatVectKey, null)$
16 $HashMapRefinement = broadcast(RDD_{FeatVect}.collect())$
17 $RDD_{Fact} = Fact$
18 $RDD_{Fact}.filter(HashMapConventional_1.hasKey(E\_Conventional_1Key)\ AND...\ AND\ HashMapConventional_m.hasKey(E\_Conventional_mKey)\ AND\ HashMapRefinement.hasKey(E\_FeatVectKey))$
19 $RDD_{Fact}.mapToPair(1, measure)$
20 $RDD_{Result} = RDD_{Fact}.reduceByKey(v_1 + v_2)$

In Algorithm 1, the inputs are the tables of the star schema of the image data warehouse, a given image s_q, the query radius r_q, and the query Q. The algorithm creates RDDs for each table present in the conventional predicate (line 1), in the similarity search predicate (lines 8 and 13), and in the extended star join (line 17). After executing the selection conditions of the conventional predicate in lines 3 and 4, the generated structures $HashMapConventional_i$ are broadcasted to all nodes of

the cluster line 5 (Figure 4a). The filtering step is carried out over the perceptual layer tables in lines 9 and 10. The candidate elements stored in the structures $HashMapFiltering_j$ are broadcasted to all nodes of the cluster in line 11. The refinement step is performed in lines 14 and 15, and the generated structure $HasMapRefinement$ is broadcasted in line 16 (Figure 4c). The extended star join operation is performed in lines 18 and 19, and the number of similar images is returned in line 20 (Figure 4d).

BrOmnImg processes each table of the star schema as an RDD. It applies the conventional and image filters on the corresponding RDDs using the operation *filter* and stores the data in hash map structures using the operations *mapToPair* and *collect*. Further, *BrOmnImg* performs the extended star join operation by applying the operations *filter*, *mapToPair* and *reduceByKey*.

4.2 $BrOmnImg^{CF}$

$BrOmnImg^{CF}$ (acronym for integrating Broadcast and Omni for processing analytical Image queries with Conventional Filter) extends *BrOmnImg* to use the conventional predicate to filter the candidate elements of the similarity search predicate before calculating the image distances in the refinement step. Contrary, *BrOmnImg* processes the conventional predicate after calculating these distances.

$BrOmnImg^{CF}$ processes the conventional predicate (Figure 5a) and the filtering step of the similarity search predicate (Figure 5b) in the same way as *BrOmnImg* does. Instead of processing the refinement step immediately after the filtering step, $BrOmnImg^{CF}$ first broadcasts each $HashMapConventional_i$ $(1 \leq i \leq m)$ and each $HashMapFiltering_j$ $(1 \leq j \leq n)$ to all nodes of the cluster and performs the extended star join over the *Fact* table in parallel. The results, i.e., every exam that satisfies the conventional predicate and the filtering step (including false positives), are stored in the structure *HashMapExtendedStarJoin* (Figure 5c). Then, $BrOmnImg^{CF}$ performs the refinement step over the *Feature Vector* table in parallel on all nodes (Figure 5d). Algorithm 2 details the algorithm of $BrOmnImg^{CF}$.

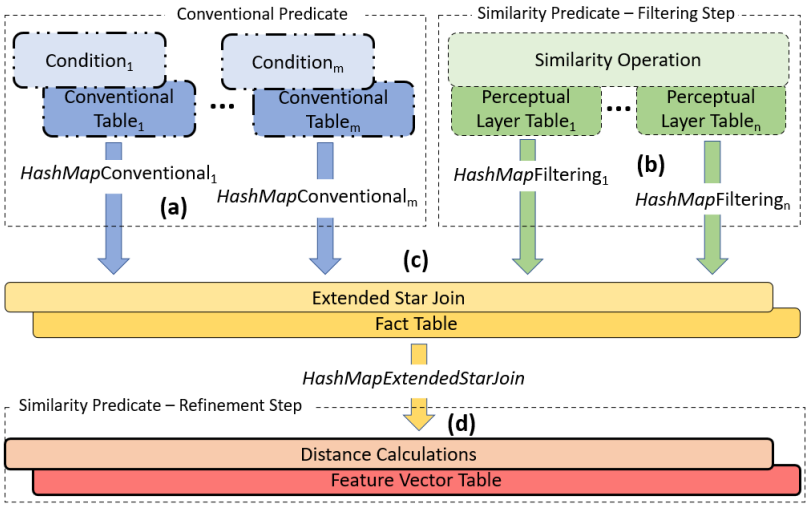

Fig. 5. General view of the proposed $BrOmnImg^{CF}$ method.

In Algorithm 2, lines 1 to 12 are the same as those of Algorithm 1 and are omitted. In lines 13 and 14, the algorithm creates the RDD for the extended star join using the structures $HashMapConventional_i$ and $HashMapFiltering_j$. In lines 15 and 16, the results are stored and broadcasted in the structure $HashMapExtendedStarJoin$. These results refer to the candidate elements of the similarity predicate

Algorithm 2: $BrOmnImg^{CF}$

Input : $Conventional_1$, ..., $Conventional_m$, $Perceptual\ Layer_1$, ..., $Perceptual\ Layer_n$, $Feature\ Vector$, $Fact$, s_q, r_q, Q
Output: Result of Q

13 $RDD_{Fact} = Fact$
14 $RDD_{Fact}.filter(HashMapConventional_1.hasKey(E\_Conventional_1Key)\ AND\ ...\ AND\ HashMapConventional_m.hasKey(E\_Conventional_mKey)\ AND\ HashMapFiltering_1.hasKey(E\_PerceptualLayer_1Key)\ AND\ ...\ AND\ HashMapFiltering_n.hasKey(E\_PerceptualLayer_nKey))$
15 $RDD_{exam}.mapToPair(E\_FeatVectKey, measure)$
16 $HashMapExtendedStarJoin = broadcast(RDD_{Fact}.collect())$
17 $RDD_{FeatVect} = Feature\ Vector$
18 $RDD_{FeatVect}.filter(if(HashMapExtendedStarJoin.hasKey(FeatVectKey))\ then\ return\ (refinementStep(FeatVectPerceptualLayer_1, s_q, r_q)\ AND\ ...\ AND\ refinementStep(FeatVectPerceptualLayer_n, s_q, r_q))$
19 $RDD_{FeatVect}.mapToPair(1, measure)$
20 $RDD_{Result} = RDD_{FeatVect}.reduceByKey(v_1 + v_2)$

filtered by the conventional predicate. In line 17, the RDD of the refinement step is created. It uses the structure $HashMapExtendedStarJoin$ to eliminate false positives by calculating the distances using the structures $HashMapFiltering_j$ (line 18). Finally, the result is calculated and returned in lines 19 and 20.

To process the refinement step after performing the extended star join operation, the $Fact$ table must have a one-to-one relationship with the $Feature\ Vector$ table. Due to this fact, $BrOmnImg^{CF}$ is more restrict than $BrOmnImg$.

5. PERFORMANCE EVALUATION

We conducted an experimental study based on a medical image data warehouse to assess the advantages of the proposed methods. The experimental setup is described in Section 5.1. The goal of our experiments was threefold: (i) investigate $BrOmnImg$ considering medical applications that require the processing of queries with very different characteristics (Section 5.2); (ii) analyze the impact of the Omni technique on $BrOmnImg$ (Section 5.3); and (iii) examine $BrOmnImg^{CF}$ with regard to the use of the conventional predicate to filter the candidate elements of the similarity search predicate (Section 5.4).

5.1 Experimental Setup

We used the *ImgDW Generator* tool [Rocha and Ciferri 2018] to populate the image data warehouse illustrated in Figure 1. The tool generated real data for the tables $ImageFeatureVector$, $ColorHistogram$, and $HaralickVariance$, and synthetic data for the remaining tables. We generated two data volumes, Volume 1 and Volume 2 (Table I). Furthermore, Table II details, for each perceptual layer, the dimensionality of the feature vector and the number of $foci$ determined by the Omni technique. $ColorHistogram$ has a high dimensionality, 256, while $HaralickVariance$ has a low dimensionality, 4. The diameter is the largest distance between any two elements of the dataset. The characteristics of the perceptual layers are independent of the data volume.

To generate different configurations, we used query $Q_{example}$ introduced in Section 2.2 as a basis and varied the selectivity of the conventional predicate, the dimensionality of the perceptual layers,

Table I. Data volumes used in the performance evaluation.

Table	Volume 1	Volume 2
Patient	300,000	2,000,000
ExamDescription	3,000,000	20,000,000
Exam	3,000,000	20,000,000
ColorHistogram	3,000,000	20,000,000
HaralickVariance	3,000,000	20,000,000
ImageFeatureVector	3,000,000	20,000,000

Table II. Characteristics of each perceptual layer considered in the performance evaluation.

Perceptual Layer	Dimensionality of the feature vector	# *foci*	Diameter of the dataset
Color Histogram	256	3	584,292.53
Haralick Variance	4	3	80.51

and the selectivity of the similarity search predicate. As the selectivity increases, the portion of data that is managed to produce the result of the query also increases. That is, low selectivity queries manage a very small portion of the data.

For the *selectivity of the conventional predicate*, we defined the following variations: (i) without a selection condition - *NotConv*; (ii) with only the selection condition $RyType$ = 'positive', determining a selectivity of 50% - *Conv50*; (iii) with only the selection condition $Gender$ = 'female', specifying a selectivity of 33% - *Conv33*; and (iv) with the selection conditions $Gender$ = 'female', $BodyPart$ = 'breast', and $ReasonInvestigation$ = 'cancer', defining a selectivity of 0.08% - *Conv0.08*.

Considering the variation of the *dimensionality of the perceptual layers*, we defined the following scenarios: (i) high dimensionality, including only $ColorHistogram$ - *His*; (ii) low dimensionality, including only $HaralickVariance$ - *Har*; and (iii) mixed dimensionality, including $ColorHistogram$ and $HaralickVariance$ - *His/Har*.

The values of *selectivity of the similarity search predicate* varied from 1% to 50%. We controlled these values by limiting the quantity of images returned by the range query similarity operation. The higher the value of the selectivity, the higher the query radius.

We defined three configurations, as detailed as follows.

—**Configuration 1.** We fixed the selectivity of the similarity search predicate to 1% and generated 12 queries by varying the values of selectivity of the conventional predicate and the dimensionality of the perceptual layers. The characteristics of the generated queries are detailed in Table III.

Table III. Characteristics of the queries of Configuration 2.

	His	Har	His/Har
NotConv	(*Q1*) NotConvHis	(*Q2*) NotConvHar	(*Q3*) NotConvHis/Har
Conv50	(*Q4*) Conv50His	(*Q5*) Conv50Har	(*Q6*) Conv50His/Har
Conv33	(*Q7*) Conv33His	(*Q8*) Conv33Har	(*Q9*) Conv33His/Har
Conv0.08	(*Q10*) Conv0.08His	(*Q11*) Conv0.08Har	(*Q12*) Conv0.08His/Har

—**Configuration 2.** We used the high dimensionality perceptual layer $ColorHistogram$ (i.e., *His*) and did not define a selection condition to the conventional predicate (i.e., *NotConv*).

—**Configuration 3.** We considered queries $Q4$, $Q7$, and $Q10$ defined in Table III. These queries refer to the high dimensionality perceptual layer $ColorHistogram$ and to the values of selectivity of the conventional predicate of 50%, 33%, and 0.08%, respectively. We also set the values of selectivity of the similarity search predicate as 1% ($Sim1$), 10% ($Sim10$), 20% ($Sim20$), 30% ($Sim30$), 40% ($Sim40$), and 50% ($Sim50$). We generated 18 queries by varying the values of selectivity of the

conventional and the similarity search predicates. The characteristics of the generated queries are detailed in Table IV.

Table IV. Characteristics of the queries of Configuration 3.

	Sim1	**Sim10**	**Sim20**	**Sim30**	**Sim40**	**Sim50**
(*Q4*) Conv50His	(*Q4Sim1*) Conv50 Sim1	(*Q4Sim10*) Conv50 Sim10	(*Q4Sim20*) Conv50 Sim20	(*Q4Sim30*) Conv50 Sim30	(*Q4Sim40*) Conv50 Sim40	(*Q4Sim50*) Conv50 Sim50
(*Q7*) Conv33His	(*Q7Sim1*) Conv33 Sim1	(*Q7Sim10*) Conv33 Sim10	(*Q7Sim20*) Conv33 Sim20	(*Q7Sim30*) Conv33 Sim30	(*Q7Sim40*) Conv33 Sim40	(*Q7Sim50*) Conv33 Sim50
(*Q10*) Conv0.08His	(*Q10Sim1*) Conv0.08 Sim1	(*Q10Sim10*) Conv0.08 Sim10	(*Q10Sim20*) Conv0.08 Sim20	(*Q10Sim30*) Conv0.08 Sim30	(*Q10Sim40*) Conv0.08 Sim40	(*Q10Sim50*) Conv0.08 Sim50

In Sections 5.2 and 5.3, *BrOmnImg* was compared with SBJ (Section 2.4). The motivations for choosing SBJ are described as follows. There is no related work in the literature that provides the same functionality as *BrOmnImg* (see Section 3). SBJ is the closest to the objective of the proposed method, since it processes the star join operation over data warehouses in Spark. In the performance evaluation of SBJ, it was compared to a wide range of algorithms available in the literature and was usually faster (between 20-50%). However, SBJ does not deal with images. Thus, we adapted it to process the similarity search predicate by using the feature vector to calculate the distances between the input image and each image of the dataset.

The experiments were performed in a cluster with 5 nodes. Each node had, at least, 3GB of RAM. We collected the elapsed time in seconds, which was recorded issuing each query 10 times, removing outliers, and calculating the average time. All cache and buffers were flushed after finishing each query.

5.2 Investigating the Performance of *BrOmnImg* for Configuration 1

In this experiment, we investigated *the variation of the selectivity of the conventional predicate and the variation of the dimensionality of the perceptual layers*. The experiment focused on medical applications that require the processing of queries with very different characteristics. Therefore, it was defined considering Configuration 1 and the characteristics of the queries described in Table III. Also, the experiment was executed considering the two data volumes detailed in Table I.

Figure 6 depicts the performance results for Volume 1. For configurations that have at least one high dimensional perceptual layer (i.e., $Q1$, $Q3$, $Q4$, $Q6$, $Q7$, $Q9$, $Q10$, and $Q12$), *BrOmnImg* provided impressive improvement in query performance that ranged from 57,32% to 64,47% when compared with SBJ. This is related to difference between the dimensionality of $ColorHistogram$ (i.e., 256) and the number of $foci$ identified for this perceptual layer (i.e., 3). This difference benefited the Omni technique to process the distance calculations, which in turn benefited *BrOmnImg*. On the other hand, for the configurations that have only the low dimensional perceptual layer (i.e., $Q2$, $Q5$, $Q8$, $Q11$), *BrOmnImg* and SBJ provided almost the same elapsed times. This is due the fact that the difference between the dimensionality of $HaralickVariance$ (i.e., 4) and the number of corresponding $foci$ (i.e., 3) was not significant.

We now comment on the performance results for Volume 2 (Figure 7). Considering the configurations that have at least one high dimensional perceptual layer (i.e., $Q1$, $Q3$, $Q4$, $Q6$, $Q7$, $Q9$, $Q10$, and $Q12$), *BrOmnImg* was from 49.91% to 65.77% faster than SBJ. As for the remaining configurations (i.e., $Q2$, $Q5$, $Q8$, $Q11$), the methods provided almost the same elapsed times.

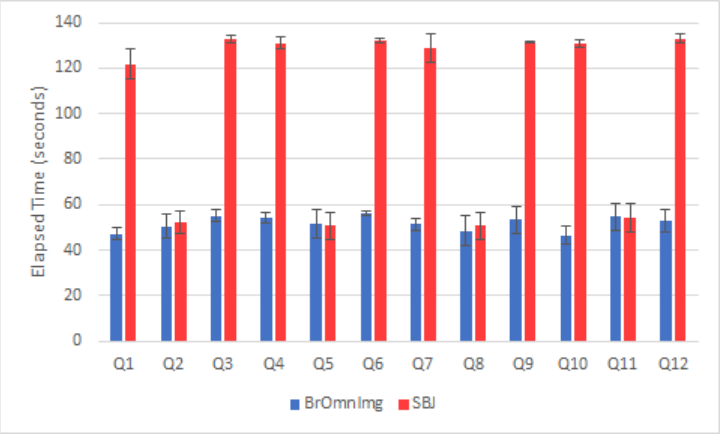

Fig. 6. Time spent by *BrOmnImg* and SBJ to process the queries of Configuration 1, considering Volume 1 and different values of the selectivity of the conventional predicate and of the selectivity of the image similarity predicate.

Analyzing the results depicted in Figures 6 and 7, we can conclude that *BrOmnImg* and SBJ showed the same trend, regardless of the data volume. Therefore, for the remainder of the performance evaluation, we will only use Volume 1 in the comparisons.

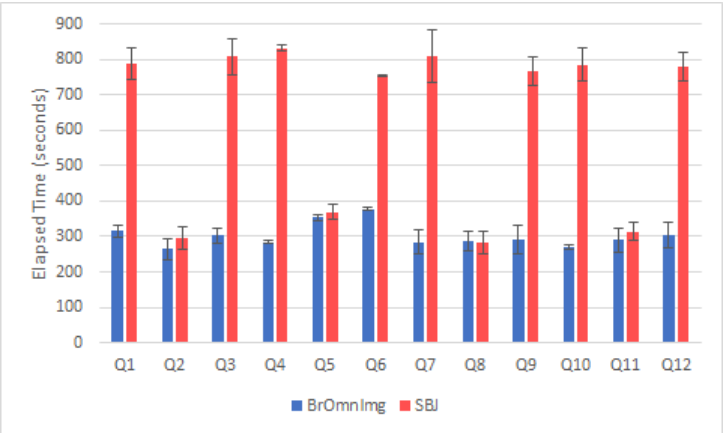

Fig. 7. Time spent by *BrOmnImg* and SBJ to process the queries of Configuration 1, considering Volume 2 and different values of the selectivity of the conventional predicate and of the selectivity of the image similarity predicate.

5.3 Investigating the Performance of *BrOmnImg* for Configuration 2

In this experiment, we investigated *the variation of the selectivity of the similarity search predicate*. The experiment was aimed to determine the effect of using the Omni technique, since the main difference between *BrOmnImg* and SBJ is the processing of the similarity search predicate. Therefore, the experiment was defined considering Configuration 2.

The performance results of *BrOmnImg* and SBJ are depicted in Figure 8, considering values of selectivity of the similarity search predicate varying from 1% (*Sim1*) to 50% (*Sim50*). For the values of 1% and 10%, *BrOmnImg* provided expressive performance gains of 56.75% and 56.94%, respectively. For the selectivity of 20%, *BrOmnImg* was 13.31% faster. Considering the values of selectivity of 30%, 40% and 50%, SBJ slightly overcame *BrOmnImg* by 1.27%, 5.40% and 7.72%, respectively. The results demonstrated that the advantage of *BrOmnImg* over SBJ decreased as the query radius

increased. This is due the fact that the increase in the value of selectivity impairs the filtering step of the Omni technique. But, this behaviour is expected, as techniques that provide prunability in searches, such as the Omni technique, are indicated to improve the performance of low selectivity queries.

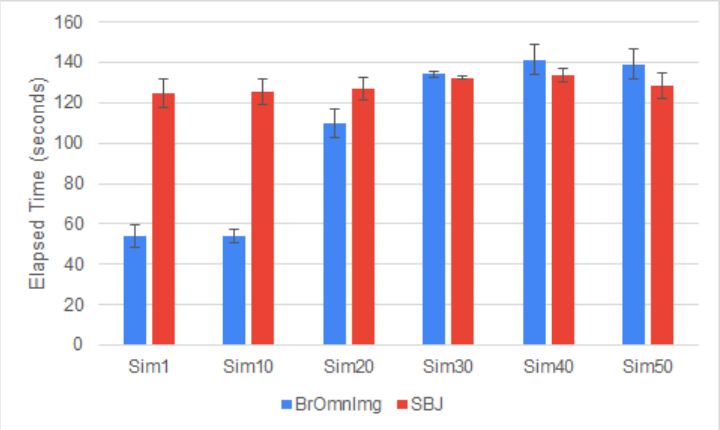

Fig. 8. Time spent by *BrOmnImg* and SBJ to process the queries of Configuration 2, considering different values of the selectivity of the similarity search predicate.

To further investigate the obtained results, we show in Figure 9 the number of distance calculations performed by each method (i.e., *BrOmnImg* and SBJ) and the number of images returned by the evaluation of the similarity search predicate. Because SBJ does not use the Omni technique, the number of distance calculations was the same as the number of stored images, i.e., 3,000,000. Considering *BrOmnImg* and the values of selectivity of 1% and 10%, the number of distance calculations was very close to the result of the similarity search predicate, benefiting the proposed method. For the remaining values of selectivity, there was a big difference between the number of distance calculations processed and the number of images returned. The performance of *BrOmnImg* decreased due to the fact that the filtering step was inefficient to filter the candidate elements plus the additional processing cost of this step.

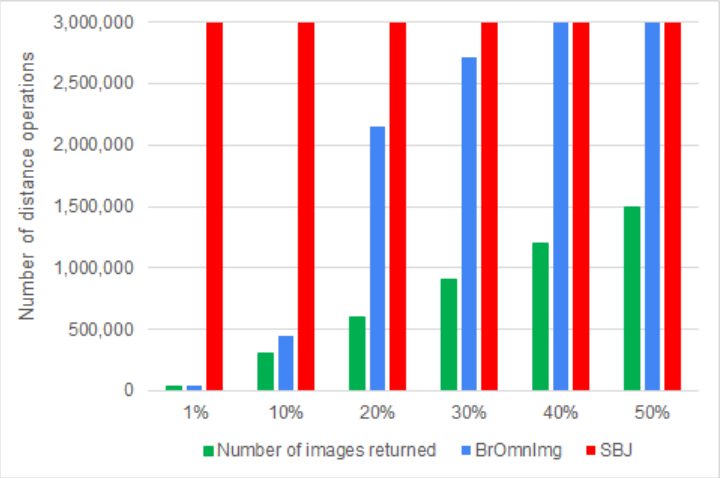

Fig. 9. Number of images returned by the evaluation of the similarity search predicate and number of distance calculations carried out by *BrOmnImg* and SBJ, considering different values of selectivity of the similarity search predicate.

5.4 Investigating the Performance of $BrOmnImg^{CF}$

$BrOmnImg^{CF}$ was compared with $BrOmnImg$ to investigate the advantages of using the conventional predicate to filter the candidate elements of the similarity search predicate before calculating the image distances. Therefore, the experiment investigated the impact of the variation of the conventional predicate, considering the high dimensionality perceptual layer $ColorHistogram$. The experiment was defined considering Configuration 3 and the characteristics of the queries described in Table IV.

The results for configurations $Q4$, $Q7$ and $Q10$ are detailed in Figs. 10a, 10b, and 10c, respectively. Regardless of the configuration, $BrOmnImg^{CF}$ and $BrOmnImg$ provided similar elapsed times for values of selectivity of the similarity search predicate of 1% and 10%. This is due to the fact that the filtering step of $BrOmnImg$ acted as a good filter for the similarity search predicate (recall Figure 9) and the improvement introduced by $BrOmnImg^{CF}$ was unable to further enhance the filter. For the remaining values of selectivity, $BrOmnImg^{CF}$ nicely overcame $BrOmnImg$. In fact, the advantage of $BrOmnImg$ over $BrOmnImg$ increased as the conventional predicate became more selective. The performance gains of $BrOmnImg^{CF}$ varied from 29.50% to 33.12% for $Q4$, from 38.01% to 44.92% for $Q7$, and from 50.24% to 54.21% for $Q10$. In these configurations, applying the selection conditions first reduced the number of candidate elements analyzed in the refinement step, benefiting $BrOmnImg^{CF}$.

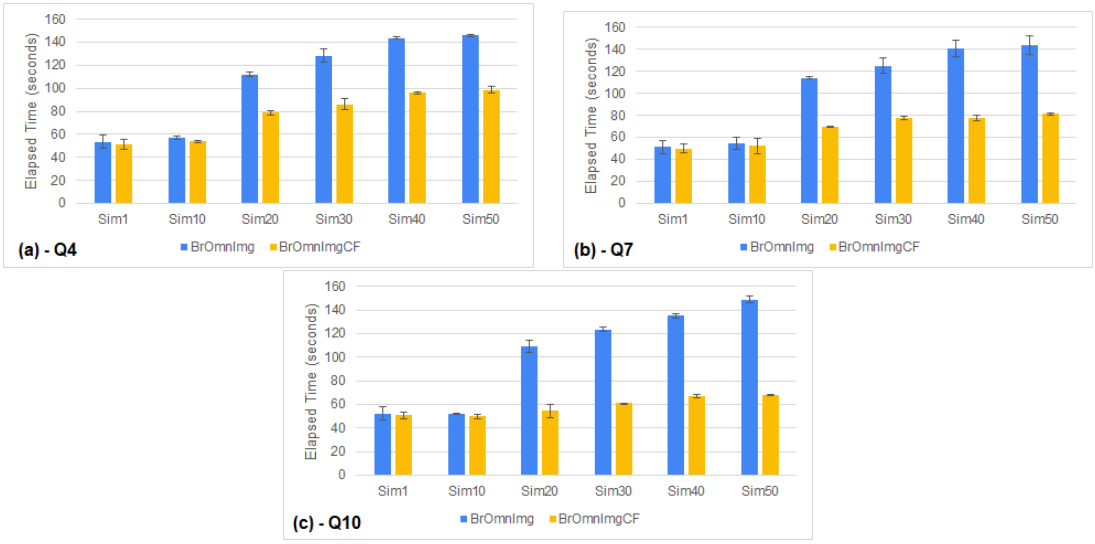

Fig. 10. Time spent by $BrOmnImg^{CF}$ and $BrOmnImg$ to process the queries of Configuration 3, considering different values of the selectivity of the conventional and similarity search predicates. (a) selectivity of the conventional predicate of 50%. (b) selectivity of the conventional predicate of 33%. (c) selectivity of the conventional predicate of 0.08%.

Figure 11 summarizes the performance results described in this section. It provides a different visualization of the results considering the values of selectivity of the similarity search predicate from 20% to 50% and the values of selectivity of the conventional predicate of 0.08%, 33%, and 50%. The difference in the performance was very similar with regard to the selectivity of the similarity search predicate, but varied with regard to the selectivity of the conventional predicate.

Comparing the performance results of $BrOmnImg^{CF}$, $BrOmnImg$, and SBJ, it is possible to highlight the following findings. For the values of selective of the similarity search predicate of 1% and 10%, $BrOmnImg$ was much faster than SBJ (Figure 8) and $BrOmnImg^{CF}$ spent almost the same elapsed time as $BrOmnImg$ (Figure 10). For the values of selectivity from 20% to 50%, the performance gain of $BrOmnImg$ over SBJ decreased as the selectivity increased, such that SBJ was slightly faster than $BrOmnImg$ for values superior to 30% (Figure 8). However, $BrOmnImg^{CF}$ impressively overcame

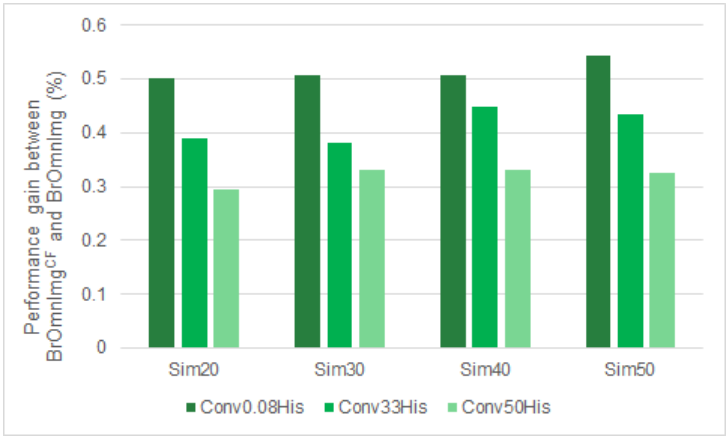

Fig. 11. Difference in the performance between $BrOmnImg^{CF}$ and $BrOmnImg$, considering different values of selectivity of the conventional and the similarity search predicates.

$BrOmnImg$ considering these values of selectivity (Figure 10). We can conclude that $BrOmnImg^{CF}$ provided better performance than SBJ, regardless of the selectivity of the similarity search predicate.

6. CONCLUSIONS AND FUTURE WORK

In this article, we focus on the efficient processing of analytical queries extended with a similarity search predicate over image data warehouses. We consider applications that manage huge volumes of data, where the use of parallel and distributed data processing frameworks is needed. To this end, we propose two methods in Spark: $BrOmnImg$ and $BrOmnImg^{CF}$.

$BrOmnImg$ integrates the broadcast join and the Omni techniques to process the star join operation and distance calculations, respectively. In the performance evaluation, we compared $BrOmnImg$ with the closest method available in the literature, SBJ. The results demonstrated that $BrOmnImg$ greatly overcame SBJ for queries composed of at least one high dimensionality perceptual layer, regardless of the selectivity of the conventional predicate. The improvement in performance varied from 49.91% to 65.77%. The results also demonstrated that $BrOmnImg$ was about 56% faster than SBJ for queries composed of a similarity search predicate with low selectivity. For the complementary scenarios, $BrOmnImg$ and SBJ spent almost the same time or SBJ was slightly faster.

$BrOmnImg^{CF}$ extends $BrOmnImg$ by using the conventional predicate to further reduce the number of distances to be calculated by the similarity search predicate. In the performance evaluation, we compared $BrOmnImg^{CF}$ with $BrOmnImg$. The results demonstrated that the methods did not provide a significant difference in performance for queries composed of a similarity search predicate with low selectivity. The complementary scenarios resulted in a performance gain of $BrOmnImg^{CF}$ over $BrOmnImg$ that ranged from 29.50% to 54.21%.

We are currently extending $BrOmnImg$ to process analytical queries extended with image, geographic, and socioeconomic similarity predicates. To comply with this extension, we are also designing different image, geographic, and socioeconomic star schemas, which may demand different query processing costs. We also plan to carry out performance tests considering real data, such as data from the COVID-19 disease. This will require efforts to generate the feature vectors and the attributes for similarity search of the images, populate the image data warehouse, and analyze the behaviour of $BrOmnImg$ and $BrOmnImg^{CF}$. Future work also includes improving the proposed methods with the use of a distributed Bitmap Join Index and with the processing on GPUs. In this article, we evaluate the performance of the proposed algorithms regarding the time spent to process queries. Another future work is to provide the time complexity analysis of the algorithms, which should consider aspects

related to the complexity of the broadcast join and the Omni techniques, as well as the investigation of the underlying characteristics of the parallel and distributed computing environment.

Acknowledgments. This work was supported by the São Paulo Research Foundation (FAPESP), the Brazilian Federal Research Agency CNPq, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Finance Code 001. G.M.R. and C.D.A.C. acknowledge support from FAPESP grants #2018/10607-3 and #2018/22277-8, respectively.

REFERENCES

Batista, N. A., Sousa, G. A., Brandão, M. A., da Silva, A. P. C., and Moro, M. M. Tie strength metrics to rank pairs of developers from GitHub. *Journal of Information and Data Management* 9 (1): 69–83, 2018.

Brito, J. J., Mosqueiro, T., Ciferri, R. R., and Ciferri, C. D. A. Faster cloud star joins with reduced disk spill and network communication. *Procedia Computer Science* vol. 80, pp. 74–85, 2016.

Brito, J. J., Mosqueiro, T., Ciferri, R. R., and Ciferri, C. D. A. Random access with a distributed bitmap join index for star joins. *Heliyon* 6 (2): e03342, 2020.

Carélo, C. C. M., Pola, I. R. V., Ciferri, R. R., Traina, A. J. M., Traina-Jr, C., and Ciferri, C. D. A. Slicing the metric space to provide quick indexing of complex data in the main memory. *Information Systems* 36 (1): 79–98, 2011.

Cuzzocrea, A. Warehousing and protecting big data: state-of-the-art-analysis, methodologies, future challenges. In *Proceedings of the International Conference on Internet of Things and Cloud Computing. Article No.: 14*. pp. 1–7, 2016.

Dash, S., Shakyawar, S., Sharma, M., and Kaushik, S. Big data in healthcare: management, analysis and future prospects. *Journal of Big Data* 6 (54): 1–25, 2019.

Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM* 51 (1): 107–113, 2008.

Fu, A. W.-C., Chan, P. M.-S., Cheung, Y.-L., and Moon, Y. S. Dynamic VP-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal* 9 (2): 154–173, 2000.

Giangreco, I., Al Kabary, I., and Schuldt, H. Adam: A system for jointly providing IR and database queries in large-scale multimedia retrieval. In *Proceedings of the 37th International ACM SIGIR Conference on Research Development in Information Retrieval*. pp. 1257–1258, 2014.

Gonzalez, R. and Woods, R. *Digital Image Processing*. Prentice-Hall, 2006.

Guoliang, Z. and Guilan, W. GBFSJ: Bloom filter star join algorithms on GPUs. In *Proceeding of the 12th International Conference on Fuzzy Systems and Knowledge Discovery*. pp. 2427–2431, 2015.

Haralick, R. Statistical and structural approaches to texture. *Proceedings of the IEEE* 67 (5): 786–804, 1979.

Hjaltason, G. R. and Samet, H. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems* 28 (4): 517–580, 2003.

Istephan, S. and Siadat, M.-R. Extensible query framework for unstructured medical data – a big data approach. In *Proceedings of the 2015 IEEE International Conference on Data Mining Workshop*. pp. 455–462, 2015.

Istephan, S. and Siadat, M.-R. Unstructured medical image query using big data – an epilepsy case study. *Journal of Biomedical Informatics* vol. 59, pp. 218–226, 2016.

Kimball, R. and Ross, M. *The data warehouse toolkit: the complete guide to dimensional modeling, 2nd Edition*. Wiley, 2002.

Kitchenham, B. and Charters, S. Guidelines for performing systematic literature reviews in software engineering, 2007.

Kuo, M., Chrimes, D., Moa, B., and Hu, W. Design and construction of a big data analytics framework for health applications. In *Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom*. pp. 631–636, 2015.

Kuo, M.-H., Sahama, T., Kushniruk, A., Borycki, E., and Grunwell, D. Health big data analytics: current perspectives, challenges and potential solutions. *International Journal of Big Data Intelligence* vol. 1, pp. 114–126, 2014.

Li, D., Zhang, W., Shen, S., and Zhang, Y. SES-LSH: Shuffle-efficient locality sensitive hashing for distributed similarity search. In *Proceedings of the 2017 IEEE International Conference on Web Services*. pp. 822–827, 2017.

Nguyen, D.-T., Yong, C. H., Pham, X.-Q., Nguyen, H.-Q., Loan, T. T. K., and Huh, E.-N. An index scheme for similarity search on cloud computing using MapReduce over docker container. In *Proceedings of the ACM International Conference on Ubiquitous Information Management and Communication*. pp. 60:1–60:6, 2016.

Nguyen, T. D. T. and Huh, E.-N. An efficient similar image search framework for large-scale data on cloud. In *Proceedings of the ACM International Conference on Ubiquitous Information Management and Communication*. pp. 65:1–65:8, 2017.

Nguyen, V.-Q., Ngoc, N., and Kim, K. Design of a platform for collecting and analyzing agricultural big data. *Journal of Digital Contents Society* vol. 18, pp. 149–158, 2017.

Raghupathi, W. and Raghupathi, V. Big data analytics in healthcare: promise and potential. *Health information science and systems* 2 (1): 3, 2014.

Raja, P. V. and Sivasankar, E. Modern framework for distributed healthcare data analytics based on Hadoop. In *Proceedings of the Second IFIP TC5/8 International Conference on Information and Communication Technology*. pp. 348–355, 2014.

Rocha, G. M. and Ciferri, C. D. A. ImgDW generator: a tool for generating data for medical image data warehouses. In *Proceedings Companion of the 33rd Brazilian Symposium on Databases: Demos and WTDBD*. pp. 23–28, 2018.

Rocha, G. M. and Ciferri, C. D. A. Processamento eficiente de consultas analíticas estendidas com predicado de similaridade em Spark. In *Proceedings of the 34th Brazilian Symposium on Databases: Short Papers*. pp. 229–234, 2019.

Sebaa, A., Chikh, F., Nouicer, A., and Tari, A. Medical big data warehouse: Architecture and system design, a case study: Improving healthcare resources distribution. *Journal of Medical Systems* 42 (4): 59, 2018.

Sebaa, A., Nouicer, A., Chikh, F., and Tari, A. Big data technologies to improve medical data warehousing. In *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*. pp. 21:1–21:5, 2017.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop distributed file system. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies*. pp. 1–10, 2010.

Tarkoma, S., Rothenberg, C. E., and Lagerspetz, E. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials* 14 (1): 131–155, 2012.

Teixeira, J. W., Annibal, L. P., Felipe, J. C., Ciferri, R. R., and Ciferri, C. D. A. A similarity-based data warehousing environment for medical images. *Computers in Biology and Medicine* vol. 66, pp. 190 – 208, 2015.

Traina-Jr, C., Filho, R. F. S., Traina, A. J. M., Vieira, M. R., and Faloutsos, C. The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *The VLDB Journal* 16 (4): 483–505, 2007.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. pp. 10–10, 2010.