SmarT: Machine Learning Approach for Efficient Filtering and Retrieval of Spatial and Temporal Data in Big Data

Sávio S. T. de Oliveira, Vagner J. S. Rodrigues, Wellington S. Martins

Universidade Federal de Goiás, Goiânia-GO, Brazil
savioteles@gmail.com, {vagner, wellington}@inf.ufg.br

Abstract. Spatiotemporal data has always been big data. In these days, big data analytics for spatiotemporal data is receiving considerable attention to allow users to analyze huge amounts of data. Traditional big data platforms cannot handle all the challenges of processing spatio-temporal data. Although some big data platforms have been proposed to process a massive volume of spatiotemporal data, neither is considered a clear winner for all possible scenarios. This paper presents the SmarT query engine, a machine learning-based solution that chooses the best big data platform for processing spatiotemporal queries on the fly. In a detailed experimental evaluation, considering the Apache Spark, Elasticsearch, and SciDB big data platforms, the response time decreased up to 22% when using SmarT.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Miscellaneous; C.2.4 [**Computer-communication networks**]: Distributed Systems—*Distributed databases*; *Distributed applications*

Keywords: Big Data, Machine Learning, Time Series Analysis

1. INTRODUCTION

Over the past few decades, time series processing has been considered one of the most challenging problems in data mining [Fawaz et al. 2019]. A time series is a collection of observations made sequentially over time. Time series arise in many domains such as finance, agronomy, health, earth monitoring, weather forecasting, to name a few. Because of advances in sensor technology, such domains may produce millions to trillions of time series, requiring fast analytical and summarizing techniques.

It all starts with the explosion in the amount of spatiotemporal data we have generated since the dawn of the digital age. Researchers want to reveal the information existing in these datasets through effective methods or techniques. Process spatiotemporal queries in a large volume of spatiotemporal data are one of the biggest challenges in the big data area [Wang et al. 2020]. However, we do not have a standardized set of big data platforms for accessing, manipulating, and performing analyses on large datasets [Wang et al. 2020; Comber and Wulder 2019].

Detailed experiments were performed in [Zhang et al. 2018] comparing big data platforms and test their effectiveness on various time series datasets. The evaluation revealed that no big data platform performs better than the others in all scenarios with spatial and temporal filters. It is up to the solution developer to choose in advance which platform to use for each query scenario. This choice is a complicated task to be made dynamically because each big data platform has characteristics that allow the best performance for certain spatial and temporal filtering scenarios.

This work presents a new query engine for big data, called SmarT(smart Spatial-Temporal query engine), for filtering and retrieving spatiotemporal data efficiently. SmarT chooses, in real time,

the best Big Data platform to process the spatiotemporal queries using a regression-based machine learning solution to support the selection of the appropriate big data platform. This work evaluated three big data platforms: Apache Spark, Elasticsearch, and SciDB. SmarT was able to decrease almost 22% of the response time compared to any of the three platforms.

The main contributions of this work are:

—A new query engine for big data, called SmarT, that chooses among multiple platforms the one most efficient to process spatiotemporal data according to the spatial and temporal query scenario.
—A new machine learning model to choose the best big data platform for spatiotemporal data processing.
—Evaluation of the solution proposed in this work with a large volume of real and synthetic data (up to 19 billion spatiotemporal objects).
—Comparison between Spark, Elasticsearch, and SciDB for spatiotemporal processing.

This paper is an extended version of [Oliveira et al. 2020], presented in the XXXV Brazilian Symposium on Databases (SBBD 2020). In particular, the present work introduces new experiments and analyses of the SmarT query engine, an updated list of related works, and a different formalization of the problem.

This paper is organized as follows. Section 2 describes the strategies found in the literature to process large volumes of spatiotemporal data. Section 3 presents the SmarT query engine and Section 4 describes the methodology and results of the tests performed. Section 5 presents the conclusions of this work and a brief description of future work.

2. RELATED WORK

There are many attempts to store spatiotemporal data in a database system. Recent studies that involve using big data frameworks like Apache Hadoop [Vavilapalli et al. 2013] and Apache Spark [Zaharia et al. 2016] platforms to store and retrieve spatiotemporal data sets have some excellent performance results on temporal and spatial analysis queries. However, most of the solutions proposed customized solutions for specific data formats or data sets and could not be applied to arbitrary multidimensional arrays. The related works in this section focus on the big data solutions for spatiotemporal storage and retrieval, and efforts done by researchers to enhance temporal and spatial data queries.

Big data is usually related to a large or complex collection of data that is difficult to handle using the present database management tools or the traditional data-processing applications. Relational database management systems and visualization packages cannot deal with big data. The limit of big data depends on the capabilities of the managing organization and in the applications used for data analysis. The relevant challenges include extraction, storage, search, sharing, transfer, analysis, and visualization of data [Tahmassebpour and Otaghvari 2016].

Due to the high computational cost and the large volume of spatiotemporal data available, some works in Big Data [de Oliveira et al. 2019; Chi et al. 2016; Comber and Wulder 2019] introduced parallel and distributed solutions for processing spatiotemporal data, particularly to support the demand for real-time processing [Ma et al. 2015]. Some researchers describe the use of Big Data in several domains of temporal and spatial data processing, such as Smart Farming [Wolfert et al. 2017; Braga et al. 2019], monitoring of water resources [Wagner et al. 2014], remote sensing image analysis [Rathore et al. 2015], IoT [Wang et al. 2015], recommendation systems [Benabderrahmane et al. 2017] and time series data mining [Fawaz et al. 2019; Amaral and de Sousa 2020; Romani et al. 2010].

With the emergence of cloud computing, users can now access data anytime, anywhere to conduct analysis on spatiotemporal data. Therefore, many distributed computing systems were built for the storage and processing of large volumes of spatiotemporal data [Almeer 2012]. The Google Earth Engine [Gorelick et al. 2017], for example, is one of the most used platforms in the processing of spatiotemporal objects, aiming to serve a wide range of scientists who do not have access to computer clusters.

The use of the MapReduce [Dean and Ghemawat 2008] programming model caused a revolution in the processing and management of spatiotemporal data [Guo et al. 2017]. MapReduce has facilitated the development of works for data mining of spatiotemporal data [Lin et al. 2013; de Assis et al. 2017; Song et al. 2015; Patterson 2011] and 3D objects [Van Den Bergh et al. 2012].

The Apache Hadoop platform [1] rapidly gained popularity among the scientific community as it successfully implemented the MapReduce paradigm, like SpatialHadoop [Eldawy and Mokbel 2015]. However, Hadoop has shown several issues when processing spatiotemporal data, and consequently, other proposals have emerged. Among them, Apache Spark [2] has been proved to be one of the most powerful big data platform. Since all the computation is done in memory in Spark, the execution of a program will be significantly faster than Hadoop (up to 100 times faster than Hadoop, according to Spark's official web page). Spark uses its fault-tolerant variables to automatically work in a distributed way, the Resilient Distributed Dataset (RDD). The optimization of in-memory processing makes Spark faster than Hadoop because disk input/output is minimized. SpatialSpark [You et al. 2015], for example, aims to provide efficient spatial operations using Apache Spark to process large scale spatial join operations.

Hadoop MapReduce and Spark are dominant Big Data processing platforms. These platforms do not provide native support for processing temporal and spatial data. Most of the solutions using big data frameworks like Hadoop and Spark were developed as customized solutions for specific data formats or spatiotemporal data sets and could not be applied to arbitrary multidimensional arrays. The array databases offer the lowest computational complexity for spatiotemporal data access. It is perfect for spatiotemporal data retrieval if indexes could be presented as integers. Popular array databases like Rasdaman [Baumann et al. 1997] and SciDB [Brown 2010; Stonebraker et al. 2011] are widely used in many practical projects, showing promising performance gain comparing to traditional methods [Xu et al. 2020]. Some works proposed new methods to index [da Silva et al. 2020] and to evaluates indexing and querying [Guedes et al. 2018] array databases.

SciDB is an open-source multi-dimensional array database. Its development was spurred by the concept that many scientific datasets have array-like structures, and there are costs to restructuring the datasets to persist as arrays within a relational database. SciDB's massively parallel processing (shared-nothing parallel database) architecture allows it to process multi-dimensional arrays on a petabytes scale. SciDB is not the only array database platform. RasdaMan is specifically designed to work with raster datasets. We have chosen SciDB in this paper because SciDB's community edition can be extended to multiple instances or nodes, whereas only the enterprise version of RasdaMan supports this. These databases have been used in several works [Lu et al. 2016; Camara et al. 2016] for efficient processing of spatiotemporal data.

ElasticSearch is also recommended for spatiotemporal queries processing because of its way of scaling and integration of spatiotemporal search queries [Shrivastava 2020]. Elasticsearch [3] is a searching and analyze engine RESTful distributed and open source. It provides a distributed, full-text search engine suitable for enterprise workloads. While not a spatiotemporal database by itself, Elasticsearch employs Lucene's column indexes, which are used to aggregate numeric values. Combined with query-

[1] hadoop.apache.org (visited on 22/05/2021)
[2] spark.apache.org (visited on 15/05/2021)
[3] https://www.elastic.co (visited on 20/05/2021)

time aggregations and the ability to index on timestamp fields, Elasticsearch provides the primitives for storing and querying spatiotemporal data. It also supports two Geo data types, including a Geo-point datatype (latitude and longitude pairs) and a Geo-shape datatype, which withstand Points, Lines, Circles, Polygons, MultiPolygons, etc. The prefix tree and geohash spatial indexing techniques support efficient spatial search.

[Zhang et al. 2018] proposed a new benchmark, named ArrayBench, used for benchmarking both Spark and SciDB. The paper also proposed the application of ArrayBench to comprehensively study the in-memory data analytics platforms (e.g., Spark), compared to the disk-oriented platforms (e.g., SciDB). In addition, ArrayBench is also used to study how the performance of scientific data analytic is correlated to various OS components, e.g., virtual memory, page cache, and distributed file systems, among others. They found that Spark does not consistently outperform SciDB in all spatial and temporal query scenarios. Unlike the proposed benchmark in [Zhang et al. 2018], our work introduces a new search engine that chooses, in real time, the best big data platform, according to the spatial and temporal query.

The most fundamental challenge for the big data platform is how to explore the large volumes and analyze the spatiotemporal data to get useful information or knowledge for future actions [Rajaraman and Ullman 2011]. The traditional big data platforms are not sufficient in dealing with these challenges [Ji et al. 2016]. Several platform solutions have been proposed in the literature containing a variety of Big Data systems for batch or real-time processing, such as Hadoop, Spark, Elasticsearch, and SciDB. The experiments carried out in some works citedoan2016evaluating,zhang2018making show that no system is more efficient than the others in all scenarios with spatial and temporal filters. Our work proposes a new query engine, called SmarT, which aims to apply the spatial and temporal filters choosing among several real-time systems the most efficient one.

3. SMART: QUERY ENGINE FOR FILTERING AND RETRIEVING SPATIAL AND TEMPORAL DATA

There are many big data platforms out created to process spatiotemporal data. Choosing one of them is a complicated task, as each system has characteristics that allow the best performance for certain spatial and temporal filtering scenarios. Moreover, the platforms are being constantly improved and each new version of them can change their ranking. This paper presents, in this section, a new query engine for filtering and retrieving spatiotemporal data, called SmarT, which chooses, in real time, the best big data platform to process a large volume of spatiotemporal data.

SmarT predicts the execution time in each big data platform and chooses the one with the lowest estimated response time, taking as input: i) the spatial and temporal query constraints; ii) the cluster metrics retrieved from the monitoring system and iii) the approximate amount of query results obtained from the indexing system. Figure 1 presents the architecture of SmarT query engine. The client sends spatiotemporal queries to SmarT using an API. This API is then responsible for sending the request to SmarT and get the query results. SmarT looks for cluster metrics like network usage, swap space, memory, and CPU in the monitoring system. The monitoring system is constantly collecting metrics from the cluster and the big data platforms. SmarT also has an indexing system for creating spatial and temporal indexes on the spatiotemporal data inserted in the big data platforms. For each query, SmarT requests cluster metrics to the monitoring system and queries the indexing system to estimate the number of results, that is, the number of observations returned to the user as a response to the query.

The data collected from the monitoring system and indexing system is the input for the machine learning algorithm that tries to predict the query processing time of each big data platform. The platform with the shortest time predicted by the algorithm process the query, filtering, and retrieving the spatiotemporal data set that satisfies the spatial and temporal constraints of the query. SmarT is

trained to minimize the response time in filtering and retrieving data by choosing the most efficient big data platform. The training data of SmarT is stored in tabular format, in which each row contains the input data and the response time of a query. Section 3.2 describes carefully the machine learning algorithm.

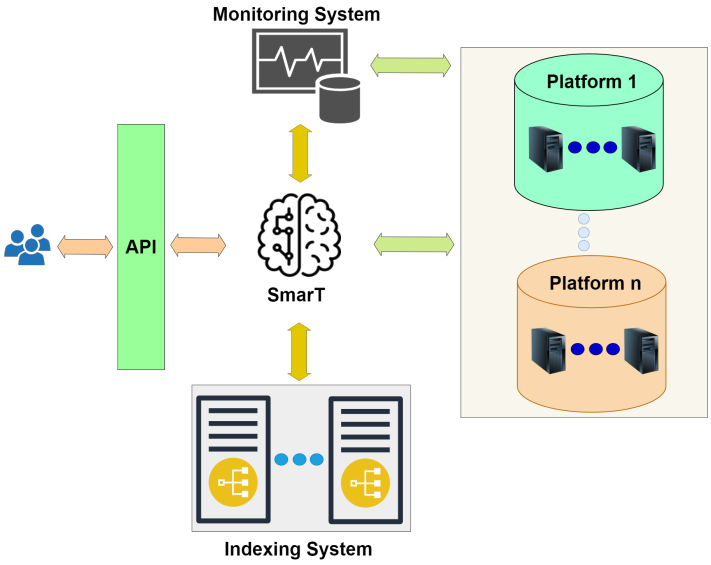


Fig. 1. SmarT architecture includes an API that interacts with users, and a central module (SmarT) responsible for choosing one of the big data platforms to run the query. SmarT makes this choice using, as input, data coming from the monitoring and indexing system, which are also part of the architecture.

3.1 Ingest and Indexing

This section provides an overview of the ingestion and indexing of objects (more details in [Oliveira 2019]). Each big data platform stores a copy of the incoming data. SmarT combines many big data platforms that together can provide the characteristics of a system that can satisfy requirements for performance, availability, and fault tolerance. SmarT is responsible for exploring the potential of these big data platforms at query time.

Many big data platforms duplicate the input data to include some or all of the following components: batch processing, online processing, stream processing, analytical data processing, and reporting [Kiran et al. 2015]. These platforms allow users to optimize the data pipeline by understanding which parts of the data need online or batch processing. However, these platforms require more skills from the developers to choose the best big data platform to run the queries and produce results [Kiran et al. 2015]. At query time, SmarT automatically tries to choose the best Big Data platform to run the query.

Each observation is composed of the following attributes: geographical point associated with the observation and its timestamp. [4] and it contains the following attributes: the geographical point associated with the observation and the timestamp of it. Therefore, the platform indexes each observation as an object with three dimensions: a) latitude, b) longitude, and c) timestamp. Observations are indexed in a distributed way to allow multiple machines or threads to index at the same time. To index the spatial objects, the spatial index *Kd-Tree* [Bentley 1975] is used, which is built on each server in the *cluster* from the spatial objects stored on the server. The k-d tree variant we implemented is the

[4]Each Big Data platform may also have its indexing system to optimize data filtering and retrieval

block k-d tree [Procopiuc et al. 2003] which is specifically designed for efficient IO, such that most of the data structure resides in on-disk blocks, with a small in-heap binary tree index structure to locate the blocks at search time. Block k-d trees are a simple yet powerful data structure. At index time, they are built by recursively partitioning the full space of N-dimensional points to be indexed into smaller and smaller rectangular cells, splitting equally along the widest ranging dimension at each step of the recursion. The block k-d tree stops recursing once there are fewer than a pre-specified (1024 in our case, by default) number of points in the cell.

A server in the indexing cluster is randomly chosen to store a new observation. This server receives the three dimensions of the observation to index it locally in the Kd-Tree. K-d trees are fast because they naturally adapt to each data set's specific distribution, using small leaf blocks where the indexed values are dense. K-d trees also naturally find the suitable trade-off how deeply to recurse by splitting up the dimensional space versus simply scanning the full-precision values in one cell is appropriate. Therefore, the result returned by Kd-Tree, in this work, is not always accurate.

The indexing system estimates the approximate number of query results. SmarT chooses any server to be the query coordinator. The coordinator routes the query to all workers holding the source indexes data. Each server returns the number of local results that meet the query constraints. The coordinator will then sum these results to get the approximate number of query results, which it then returns to the SmarT.

3.2 Using Machine Learning for Real-Time Selection of Big Data Platform for Spatio-Temporal Queries

This section presents our solution to reduce the response time when executing queries with spatial and temporal filters in a distributed environment trying to choose, in real time, the best big data platform for query processing. Since the response time is a continuous variable, the predictive regression modeling tries to solve the problem of minimizing the response time, aiming to approximate the mapping function to the continuous output variable that represents the response time. The regression algorithm tries to predict the response time of each platform and chooses the one with the least expected time. Even if the best platform is not chosen, it increases the possibility of choosing one of the best ones when the problem is modelled as a regression task.

The training of the machine learning model is done for each big data platform, where the dependent variable is the response time of the respective platform and the independent variables are the query constraints, monitoring metrics, and the estimated number of query results. We will denote the dependent variable by y and the set of independent variables by $x_1, x_2, ..., x_p$, where p indicates the number of independent variables. The relationship between y and $x_1, x_2, ..., x_p$ is approximated by the regression model of equation 1:

$$y = f(x_1, x_2, ..., x_p) + \varepsilon \tag{1}$$

where $f(x_1, x_2, ..., x_p)$ provides an acceptable approximation to the true relation between y and $x_1, x_2, ..., x_p$, and ε measures the discrepancy in the approximation.

There are many regression models [Chatterjee and Hadi 2015] used to estimate the response time of each system. Ensemble learning methods are the best technique when the performance on a predictive modeling project is the most important outcome [Zhou 2012; Polikar 2006]. Ensemble learning helps to improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model [Opitz and Maclin 1999].

Our solution explores the ensemble XGBoost (Extreme Gradient Boosting) [Chen and Guestrin 2016] method. Since its introduction in 2014, XGBoost has quickly become among the most popular methods used on Kaggle. It has accumulated an impressive track remind of winning contest [Nielsen

2016], most of them with tabular input data (which is precisely our input format). Uncovering some possible reasons why tree boosting is so effective is interesting for many reasons. First, it might improve understanding of the inner workings of tree boosting methods. Second, it can aid in further development and improvement of the current tree boosting methodology. Third, by understanding the core principles of tree boosting which makes it so versatile, we might be able to construct whole new learning methods which incorporate the same core principles [Nielsen 2016].

XGBoost is one of the most popular and efficient implementations of the Gradient Boosted Trees algorithm, a supervised learning method that is based on function approximation by optimizing specific loss functions as well as applying several regularization techniques. The following pairs of training examples are input for the algorithm: $(\vec{x}_1, y_1)$, $(\vec{x}_2, y_2)$, $(\vec{x}_p, y_p)$, where $\vec{x}$ is a vector of characteristics containing the independent variables and y is the response time of the query. XGBoost includes a regularization term that controls the complexity of the model, which helps to avoid overfitting [Hawkins 2004] and to support arbitrary cost functions. The objective function presented in equation 2 is built with two parts, the first being a training loss over the training set and the second part the regularization term that penalizes the model's complexity:

$$Obj = \sum_i L(y_i, \hat{y_i}) + \sum_k \Omega(f_k) \tag{2}$$

where $L(y_i, \hat{y_i})$ is the training loss function, and $\Omega(f_k)$ is the regularization term. The training loss measures how predictive our model is regarding the training data. $\Omega(f_k)$ describes the complexity of the tree f_k and is defined in the XGBoost algorithm by equation 3:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda w^2 \tag{3}$$

where T is the number of leaves in the tree f_k and w the weights of the leaves (i.e., the predicted values stored in the leaf nodes). When $\Omega(f_k)$ is included in the objective function, there is a need to optimize the model to build less complex trees while minimizing $L(y_i, \hat{y_i})$. This helps to reduce the *overfitting*. A penalty is applied, with γT, for each additional new leaf in the tree. Leaf weights with values too high or too low are penalized with λw^2. Both γ and λ are user-configurable values.

Given that the boosting method runs iteratively, it is possible to define the objective function for the current iteration m in terms of the prediction from previous iterations, $\hat{y_i}^{(m-1)}$, fitted by the decision tree f_k, as can be seen in equation 4:

$$Obj^m = \sum_i L(y_i, \hat{y_i}^{(m-1)} + f_k(x_i)) + \sum_k \Omega(f_k) \tag{4}$$

The input data for XGBoost can be described as a matrix where each row represents an instance and each column a feature. In our work, the following features are used:

(1) area: spatial filter area;
(2) time_interval: filter time interval in days;
(3) count: estimated number of query results;
(4) swap_free: the average of available swap area on servers in the cluster;
(5) mem_free: the average of available physical memory on servers in the cluster;
(6) load_one: system load average over the last 1 minutes on servers in the cluster;
(7) load_five: system load average over the last 5 minutes on servers in the cluster;
(8) load_fifteen: system load average over the last 15 minutes on servers in the cluster;

(9) cpu_user: CPU usage average by user processes on servers in the cluster;
(10) cpu_system: CPU usage average by kernel processes on servers in the cluster;
(11) bytes_in: incoming network traffic average on servers in the cluster;
(12) bytes_out: outgoing network traffic average on servers in the cluster.

Since the response time is the dependent variable y, then for each training input pair $(\vec{x}_i, y_i)$, the dependent variable y_i follows the equation 5 and the feature vector $\vec{x}_i$ by equation 6:

$$y_i = response\_time_i \tag{5}$$

$$\begin{aligned} \vec{x}_i = (&area_i, time\_interval_i, count_i, swap\_free_i, mem\_free_i, \\ &load\_one_i, load\_five_i, load\_fifteen_i, \\ &cpu\_user_i, cpu\_system_i, bytes\_in_i, bytes\_out_i) \end{aligned} \tag{6}$$

The independent variables *area* and *time_ interval* are obtained from the spatial and temporal filters set in the query by the user. These variables are important because they have a significant impact on the computational cost of queries. They directly influence the number of returned results since, usually, the higher the spatial and temporal constraint, the higher the number of results returned. However, this statement may not be true when there is no data in the dataset that meets the query constraint. For example, for a database of Brazil, queries covering the entire African continent will return no results. Small spatial filtering within Brazil, on the other hand, in this database, may return some set of results.

The variable *count* is the estimated number of results given the spatial and temporal restrictions of the query returned by the indexing system. By combining the *area*, *time_ interval* and *count* variables, it is possible to more accurately estimate the computational cost of the query. This information is important for SmarT's decision of which system to run the query, since each system may perform differently according to the volume of data to be retrieved.

The cluster metrics get from the monitoring system are important because they impact the performance of a big data platform in a distributed environment. The average of each monitoring metric shows the cluster state at query run time. Although the *area*, *time_ interval*, and *count* variables allow us to estimate the impact of the cost of filtering and retrieving query data, the cluster metrics like memory, disk, CPU, and network usage can also impact the choice made by SmarT. Each system may have internal optimizations in which one or more cluster usage metrics can influence system performance. Apache Spark, for example, is an in-memory query processing platform, so the metrics *memory-free* and *swap-free* directly affect its performance.

4. EVALUATION

This section evaluates SmarT's ability to choose the best Big Data platform for each query scenario. Three platforms will be evaluated: i) Elasticsearch version 7.0.1, ii) SciDB version 18.3, and iii) Apache Spark version 2.3.2. The hypothesis is that none of these three platforms can be the most efficient in all query scenarios with spatial and temporal filters.

4.1 Experimental Setup

The tests were performed on a cluster with 8 servers in the Amazon EC2 [5] Cloud Computing environment. The three platforms (Elasticsearch, Spark, and SciDB) were configured in the 8 servers.

[5]https://aws.amazon.com/pt/ec2/ (visited on 10/01/2021)

The computing nodes hold the following characteristics: Intel® Xeon® Platinum 8175M, 16 vcores, 3.1 GHz, 128 GB RAM, and 600 GB SSD. The servers were connected by a 10 Gbit/second Ethernet network with dedicated bandwidth of 3500 Mbps. The monitoring and indexing system was also deployed on these 8 machines.

The experiments aimed to measure the response time of each platform to filter and retrieve the query results. This analysis is important to verify if SmarT can reduce the query response time. To perform this measurement, a server sent requests to the cluster and collects the metrics. This node holds the following characteristics: 2.4 GHz Intel Xeon E5-2676 v3 processor, 8 cores, 32 GB of RAM, and 1 TB SSD.

More than 7.2 billion observations related to 16.7 million pixels of an area of 800.824 km^2 covering a piece of the Midwest and Southeast of Brazil, as can be seen in Figure 2, have been stored on every big data platform. Each pixel has a time series with 435 observations between 2000 and 2019 extracted from NASA's MOD13Q1 product [6] (National Aeronautics and Space Administration), with a periodicity of 16 days between observations and a spatial resolution of 250 meters.

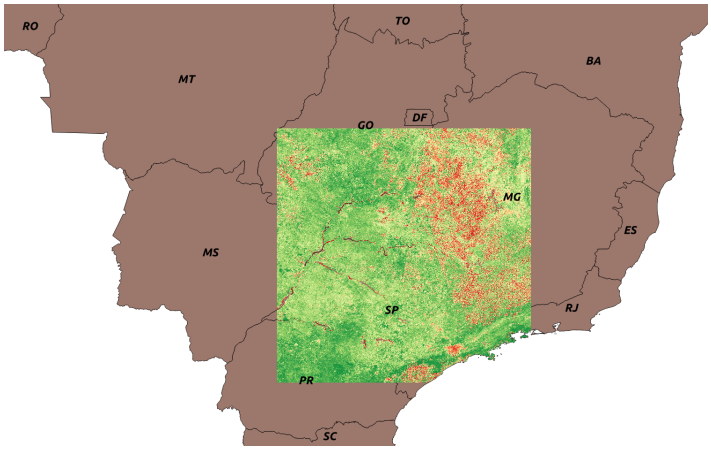


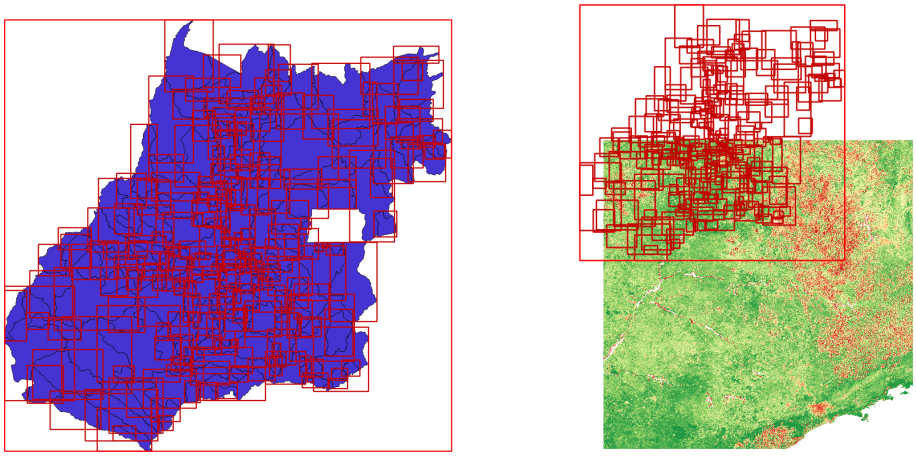
Fig. 2. This input dataset has an area of 800.824 square kilometers covering a piece of the Midwest and Southeast of Brazil.

Queries were executed with spatial and temporal constraints to evaluate the performance of each of the three platforms and the SmarT to choose the best one according to the query. The spatial filter was taken from the MBR (Minimum Bounding Rectangle) of the geometry that delimits the State of Goiás in Brazil, as can be seen in Figure 3(a). The MBR of a geometry is the bounding geometry formed by the minimum and maximum (X, Y) coordinates [Casanova et al. 2005]. These geographic areas allow evaluating many spatial restriction scenarios (Figure 3(b)), filtering from areas that have no results in the query as well as areas that return more than 2 billion of results, as in the case of the MBR of the State of Goiás.

The temporal restrictions allow for an evaluation of the performance of the big data platforms in applying the temporal filter on the data. Table I shows the time ranges defined as temporal restrictions in the database queries and the number of objects returned when filtering the database using only the temporal axis. Using the spatial and temporal filters at the same time, queries return from 0 to 333 million results.

In all, 1983 query constraints were generated from 247 spatial filters, 7 temporal filters, and 1729 (247×7) spatiotemporal filters. For each of the 1983 constraints, we ran two queries. An example of

[6]This dataset is available by LAPIG (Image Processing and Geoprocessing Laboratory)



(a) MBRs for each city and for the entire State of Goiás in Brazil that are used in the spatial filtering.

(b) MBRs filter out a chunk of the evaluation area allowing them to evaluate many test scenarios concerning to the number of query results.

Fig. 3. Map with the MBRs of the cities of the State of Goiás used in the queries.

a spatial query could be to find the objects that match the spatial filter at any time. The temporal filter tries to find the objects of the temporal filter in any geographical place. To return the objects that match a spatiotemporal filter, we try to find those that satisfy the spatial and temporal filter.

The first query aims to analyze the performance of each platform to filter without retrieving the results. The second query aims to analyze the performance of each platform to filter and retrieve the query results. We executed each query 10 times, collecting response times and measurements of CPU, network, memory, and disk. These measurements were collected and used to train SmarT to choose the best platform according to the query filters. In total, 118980 queries were executed, built from 1983 constraints, with two queries for each constraint, being executed 10 times on each of the three platforms ($1983 \times 2 \times 10 \times 3$).

Table I. Time filter constraints and the approximated number of query results just with temporal filtering.

Time filtering range	**Approximate number of query results**
2016-01-01 to 2016-05-24	167 million
2016-01-01 to 2016-10-31	335 million
2016-01-01 to 2017-04-07	503 million
2016-01-01 to 2017-09-14	671 million
2016-01-01 to 2018-02-18	839 million
2016-01-01 to 2018-07-28	1 billion
2016-01-01 to 2019-01-01	1.17 billion

SciDB and Apache Spark were installed on the cluster with the default configuration, without any optimization or customization. As for the Elasticsearch, the default value of 1 GB for JVM heap memory was changed [7] to 48 GB, as queries were causing the heap memory to overflow during testing. Evaluation of Spark was performed using the Hadoop ecosystem version 3.1.1, with Apache Yarn scheduling the queries and managing the cluster resources required by Spark jobs, and HDFS as the storage system. The data were stored in HDFS in Parquet format with Snappy [8] compression, which,

[7] Elasticsearch was developed in Java.
[8] https://github.com/google/snappy (visited on 22/02/2021)

compared to CSV format, reduces query response time up to 34 times and uses up to 87% less disk space [9]. Spark, SciDB, and Elasticsearch have their storage layer, replicating the spatiotemporal data on each platform.

4.2 Evaluation of the Big Data platforms: Apache Spark, SciDB and Elasticsearch

This section evaluates the performance of Elasticsearch, Apache Spark, and SciDB in spatial and temporal filtering over the remote sensing time series. In Section 4.3, we will compare the performance of SmarT against the platforms to define if there was any positive impact on response time. Table II shows the query response time of each platform for filtering without retrieving query results. Elasticsearch had the lowest average response time due to its efficient temporal and spatial indexing framework. Apache Spark averaged better response time than SciDB, but with a higher standard deviation, as Spark is better for queries returning between 100.000 and 500.000 results, but worse for queries with more than 500.000 results.

Table II. Queries response time, in milliseconds, to filter without retrieving the results on the three big data platforms.

	Apache Spark	Elasticsearch	SciDB
Average response time (ms)	321	4	398
Standard deviation of response time (ms)	697	12	332

Table III shows the average response time to filter and retrieve the query result. Elasticsearch had the worst performance among the three platforms, being more than ten times worse than SciDB and more than seven times worse than Spark. This difference in performance, seen in Tables II and III, is explained by the fact that Elasticsearch has a very efficient internal indexing mechanism, but its performance depends on the size of the dataset returned from the database [10]. When the result retrieved gets larger than a certain threshold [11], the response time increases considerably [Thacker et al. 2016], so Elasticsearch starts to perform worse than the other two platforms.

Table III. Analysis of the queries response time (ms) in the platforms.

	Apache Spark	Elasticsearch	SciDB
Average response time (ms)	822	6291	605
Standard deviation of response time (ms)	1842	9295	430
Maximum response time (ms)	49986	78574	2946

Table IV compares the percentage of times that each of the three platforms was the best choice using three ranges of query results: i) up to 100.000, ii) between 100.000 and 500.000, and iii) more than 500.000. For few results (less than 100.000), Elasticsearch is the best choice among the three platforms, but above 100.000 returned results, SciDB and Spark become better options. This can be explained by the Elasticsearch average time in Table III, as it has a worse response time for queries that return larger amounts of data, and as they have longer response time, it impacts the average response time of the three platforms.

Spark performs better than SciDB when it is not necessary to return a high volume of data. This is because Spark has to go through all the Parquet files to filter the results, and when the database is larger than the memory, the intermediate results need to be serialized as Java objects and stored on secondary storage devices (disk or SSD) or recomputed. This serialization and deserialization overhead is critical for Spark for large volumes of data retrieved from disk and returned in the query

[9] https://dzone.com/articles/how-to-be-a-hero-with-powerful-parquet-google-and (visited on 01/03/2021)
[10] https://www.elastic.co/guide/en/elasticsearch/reference/current/general-recommendations.html (visited on 10/01/2021)
[11] This threshold depends heavily on the Elasticsearch configuration and the *hardware* infrastructure of the **cluster**.

result. SciDB, on the other hand, has an optimization for I/O accesses, minimizing file access overhead that allows it, for large volumes of data returned, to perform better than Spark and Elasticsearch. SciDB is a platform designed to handle time series data, unlike Spark, which is a general processing engine for large volumes of data. For a large volume of multidimensional data, SciDB can outperform Spark [Zhang et al. 2016] taking advantage of its efficient storage, indexing, and retrieval system.

Table IV. Percentage of times that each of the three platforms was the best choice using three ranges for evaluating result retrieval: i) up to 100.000 results, ii) between 100.000 and 500.000 results, and iii) more than 500.000 results.

	Up to 100.000	Between 100.000 and 500.000	Over 500.000
Elasticsearch	90%	0%	0%
SciDB	1%	22%	64%
Apache Spark	9%	78%	36%

Tables II and III show that the difference between SciDB and Spark response time for filtering and retrieving time series data were not as significant as Elasticsearch. SciDB has an optimized internal structure that can maintain a low standard deviation in response time with a maximum response time of 2946 milliseconds. Spark, on the other hand, with its Parquet format data, is more optimized than Elasticsearch but has a longer maximum response time (49986 milliseconds).

Comparing the three platforms, Elasticsearch is recommended for a smaller query result size, Spark for a medium query result size, and SciDB for a larger query result size. The challenge, however, is the choice of the big data platform, as the definition of a smaller, medium, or larger result set is subjective and depends on the *hardware* configuration of the cluster, the available platforms, and the spatial and temporal filtering scenarios.

4.3 Evalutation of SmarT

This section presents the results and discussion regarding SmarT in choosing the platform for query execution in each scenario. It is possible to see in Table IV, that no platform completely dominates the others by analyzing the percentage of times each had the shortest response time in each range of query result size. This leads to the elaboration of a hypothesis that other features also impact the response time of the platforms. This paper proposes a query engine, SmarT, that uses machine learning to automatically choose the best platform using some machine learning features available: *area*, *time_ interval*, *count*, *swap_ free*, *mem_ free*, *load_ one*, *load_ five*, *load_ fifteen*, *cpu_ user*, *cpu_ system*, *bytes_ in*, *bytes_ out*.

The features *area* and *time_ interval* are calculated at query time from the spatial and temporal constraints, respectively. The feature area has been normalized by multiplying the value by 100, and for cases where there is no spatial filter, the value is zero. The feature interval was discretized, creating eight categories, seven of them being the ones shown in table I and the last category the case where there is no temporal filter.

The variable *count* comes from an indexing service that returns the estimated number of query results given the spatial and temporal constraints. This variable was discretized into 15 categories with equal interval width according to the count value. The remaining features come from cluster metrics collected by the monitoring system using the Ambari [12] framework. These variables are described in detail in Section 3. The cpu and load variables were normalized multiplying the value by 100 and rounding the value to two decimal places. The network and memory variables were normalized by dividing the values by 1000 and rounding the result to two decimal places.

[12]https://ambari.apache.org/ (visited on 22/05/2021)

The supervised machine learning algorithm XGBoost [Chen and Guestrin 2016] was implemented using the *open-source* H2O [13] platform, version 3.24.0.4. The experiments were performed using the cross-validation method called k-folds, with k=5, using the default values ($\gamma = 0$ and $\lambda = 1$) for the parameters of the XGBoost algorithm in the H2O platform. The measurements collected from 118.980 queries were used as training samples, extracting 75% for the training set and 25% for the test set.

SmarT achieved an overall accuracy of 90.1% in choosing the best platform, and even in the cases where it does not choose the best platform, there is a small difference in response time if it had always chosen the best one for each query. In 95% of the cases, the platform chosen by SmarT is either the best or has response times up to 10% longer than the time of the best platform. As it can be seen in Table V, using SmarT, the query response time was 496ms versus 482ms if it had always chosen the best platform ("Best" column). We have measured the model error in predicting quantitative data using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). SmarT had an RMSE equal to 80.51231 and MAE equal to 47.88053.

Table V. Comparison of the average response time (ms) between SmarT and the three big data platforms. The "Best" column indicates the average response time if the best option was always chosen by SmarT.

	SmarT	Best	Apache Spark	Elasticsearch	SciDB
Elasticsearch + SciDB + Spark	496	482	822	6291	605
Elasticsearch + SciDB	537	533	∅	6291	605
Elasticsearch + Spark	800	796	822	6291	∅
Spark + SciDB	527	513	822	∅	605

The average response time of SmarT is calculated from the average response times of the chosen platform on each query. To understand the calculation of this average time, suppose that for a given query, SmarT predicts as response times: 434ms for Elasticsearch, 390ms for Apache Spark, and 1023ms for SciDB. Indeed, the response times were 455ms, 460ms, and 840ms, respectively. In this case, SmarT chose Apache Spark to execute the query and add 460ms for the final average time calculation. For the "Best" response time average, it would add the value of 455ms, as it was the shortest time among the three platforms. For each query, the time to collect the cluster metrics, compute the area and time interval, and query the indexing system to estimate the number of query results is added to the final SmarT time. Table V shows the SmarT results including this overhead (on average 3ms).

The variable importances were computed from the gains of their respective loss functions during tree construction, using a squared error based on gradient and hessian. To estimate the response time of the platforms using XGBoost, the most important feature in SmarT's decision making was *count*, as was already expected by the results shown in section 4.2. To estimate the Elasticsearch query response time, other features had an impact, in order of priority: *area*, *time_interval* and *bytes_out*. The variables *area* and *time_interval* are related to the number of returned results, since generally, the greater the spatial and temporal constraint the greater the number of returned results. The *bytes_out* feature had an impact on Elasticsearch due to the traffic cost of the query results, inherent to Elasticsearch which does not have major optimizations to reduce the network traffic, like Spark and SciDB.

To estimate the Spark query response time, in addition to the *count*, *area* and *time_interval* features, the amount of free memory (feature *mem_free*) impacted the prediction of the machine learning model. This is because Spark does query processing using in-memory computation and is therefore impacted by the amount of free memory at query time. In some cases, Spark has to use the *swap* area of the operating system when there is not enough memory.

[13] https://www.h2o.ai/ (visited on 08/01/2021)

To estimate the query response time in SciDB, in addition to the *count*, *area* and *time_interval* features, the variables *load_one*, *load_five* and *cpu_system* had an impact to predict the response time. SciDB has an optimized internal indexing and storage mechanism for retrieving data from disk, but keeping this mechanism efficient requires high processing cost and CPU usage. Therefore, the usage of the cluster impacts on SciDB's response time.

4.4 Discussion

Table V also presents the comparison of the average response time of SmarT against Spark, Elasticsearch, and SciDB. In this table, a comparison of the response time is also provided when SmarT has all platforms or only two of them to choose from. SmarT had an average response time close to "Best" with a difference of less than 3% for all platform combinations. With two platforms, Spark and SciDB had the best performance, followed by Elasticsearch and SciDB with a small average time difference of 10 ms between the two. The duo Elasticsearch and Spark had the worst performance, being 33% worse than the other options. As expected, from the evaluation of Table II, SciDB is the most important platform among the three for reducing response time. The average time for all queries in SciDB is the lowest among the three because it has the best performance when the query returns a high number of results. For these cases, the times in Elasticsearch and Spark get higher, as can be seen in Table IV, which impacts the final average time.

It is recommended to have all three platforms to have the best possible performance. But if it is not possible, analyzing the results of Table V and Table IV, the pair Elasticsearch and SciDB should be used if there is a scenario in which there will be many queries with constraints that return few results. When the spatial and temporal filtering constraints are not known, or in query scenarios with a large volume of results, it is ideal to use the pair Spark and SciDB.

When choosing only one big data platform, SciDB proved to perform better than Spark and Elasticsearch in filtering and retrieving time series data. But it was evidenced in Table V that using SmarT with all three platforms is the better choice. SciDB had a 21.9% higher average response time than SmarT when all platforms are available to SmarT. Even when pairing Elasticsearch with SciDB and Spark with SciDB using SmarT to choose the best in real time, SciDB had an average response time of 12.6% and 14.8% longer than the pair, respectively.

It is usual in big data architectures, such as Lambda [Kiran et al. 2015] and Kappa architectures [Lin 2017], to have more than one big data platform in the cluster to handle different query scenarios [Feick et al. 2018]. Thus, the SmarT query engine, with 90.1% accuracy and almost 22% response time reduction over SciDB, proved to be a good choice in big data architectures to process spatial and temporal queries over large volumes of time series data.

5. CONCLUSION

The large volume of unstructured and high dimensional data poses the problem of processing time series in the context of Big Data [Fan et al. 2014]. The efficient processing of a large volume of spatiotemporal data is one of the biggest open challenges in the area. This work proposes an intelligent engine, called SmarT, that adapts itself to the environment of available systems automatically, in real time, being capable of efficiently filtering and retrieving time series data. SmarT uses a machine learning algorithm to choose the best Big Data system, among several existing ones, to filter and retrieve the query data in the shortest possible time.

The experiments reported in this work showed SmarT achieving more than 90% accuracy in choosing the best system between Apache Spark, Elasticsearch, and SciDB. As a result, there was a 22% reduction in response time using SmarT than if only one of the three systems processes the query. It would be practically impossible to make this choice manually, following each query, due to the

number of possible combinations of variables considered. SmarT represents an important step towards time series processing research, as it allows researchers to focus on building time series processing algorithms, without worrying about the challenge of efficiently filtering and retrieving large volumes of data.

SmarT training is carried out periodically by executing a set of queries on the existing systems and collecting the metrics to be used as variables of the machine learning methods. During the collection of metrics, it is not recommended running other queries in the cluster to not negatively impact the response time of user's queries. The collection of metrics for SmarT should be performed at times of the day when there is historically little use of the cluster. In this case, SmarT is limited to always being trained with the same set of queries and in predetermined periods. To generate a training set that better represents the scenarios of queries made by users, we intend to modify SmarT to generate the training set from real user queries, training the model in periods of idleness of the cluster automatically. Other variables will be added to the predictor, such as the data partitioning criterion that profoundly impacts big data processing.

Another limitation of SmarT is that data must be replicated among the systems, as there is no efficient bus for data storage and retrieval. In future work, we will carry out cost-benefit assessments in replicating the data on various platforms and measuring the gain in response time. In addition, we plan to create an efficient data bus that is capable of integrating with the various big data systems. A challenge of this bus is to have the least possible impact on latency with the replacement of the native storage layer of each system. Another challenge is to build a data bus that facilitates integration with other systems that may be added. Lastly, we also intend to evaluate other machine learning models, in addition to XGBoost, and the impact of other variables, such as internal metrics for each big data platform.

Acknowledgement

This work was funded by Companhia Paranaense de Energia (Copel), P&D ANEEL-Copel Distribuição, project number 2866-04842017.

REFERENCES

Almeer, M. H. Cloud hadoop map reduce for remote sensing image analysis. *Journal of Emerging Trends in Computing and Information Sciences* 3 (4): 637–644, 2012.

Amaral, T. and de Sousa, E. P. M. Mining temporal exception rules from multivariate time series using a new support measure. *Journal of Information and Data Management* 11 (3), 2020.

Baumann, P., Furtado, P., Ritsch, R., and Widmann, N. The rasdaman approach to multidimensional database management. In *Symposium on Applied Computing: Proceedings of the 1997 ACM symposium on Applied computing*. Vol. 1997. pp. 166–173, 1997.

Benabderrahmane, S., Mellouli, N., Lamolle, M., and Paroubek, P. Smart4job: A big data framework for intelligent job offers broadcasting using time series forecasting and semantic classification. *Big Data Research* vol. 7, pp. 16–30, 2017.

Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18 (9): 509–517, 1975.

Braga, D. J. F., da Silva, T. L. C., Rocha, A., Coutinho, G., Magalhães, R. P., Guerra, P. T., de Macêdo, J. A., and Barbosa, S. D. Time series forecasting to support irrigation management. *Journal of Information and Data Management* 10 (2): 66–80, 2019.

Brown, P. G. Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, pp. 963–968, 2010.

Camara, G., Assis, L. F., Ribeiro, G., Ferreira, K. R., Llapa, E., and Vinhas, L. Big earth observation data analytics: matching requirements to system architectures. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. ACM, pp. 1–6, 2016.

Casanova, M. A., Câmara, G., Davis, C., Vinhas, L., and Queiroz, G. R. *Banco de dados geográficos*. Mundo-GEO Curitiba, 2005.

Chatterjee, S. and Hadi, A. S. *Regression analysis by example*. John Wiley & Sons, 2015.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp. 785–794, 2016.

Chi, M., Plaza, A., Benediktsson, J. A., Sun, Z., Shen, J., and Zhu, Y. Big data for remote sensing: Challenges and opportunities. *Proceedings of the IEEE* 104 (11): 2207–2219, 2016.

Comber, A. and Wulder, M. Considering spatiotemporal processes in big data analysis: Insights from remote sensing of land cover and land use. *Transactions in GIS*, 2019.

da Silva, A. C., Lustosa, H. L. S., da Silva, D. N. R., Porto, F. A. M., and Valduriez, P. Savime: An array dbms for simulation analysis and ml models prediction. *Journal of Information and Data Management* 11 (3), 2020.

de Assis, L. F. F. G., de Queiroz, G. R., Ferreira, K. R., Vinhas, L., Llapa, E., Sanchez, A. I., Maus, V., and Câmara, G. Big data streaming for remote sensing time series analytics using mapreduce. *Revista Brasileira de Cartografia* 69 (5), 2017.

de Oliveira, S. S. T., Martins, W. S., Sacramento, V., Bueno, E., Cardoso, M., and Pascoal, L. A parallel and distributed approach to the analysis of time series on remote sensing big data. *Journal of Information and Data Management* 10 (1): 16–34, 2019.

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

Eldawy, A. and Mokbel, M. F. Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering*. IEEE, pp. 1352–1363, 2015.

Fan, J., Han, F., and Liu, H. Challenges of big data analysis. *National science review* 1 (2): 293–314, 2014.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery* 33 (4): 917–963, 2019.

Feick, M., Kleer, N., and Kohn, M. Fundamentals of real-time data processing architectures lambda and kappa. *SKILL 2018-Studierendenkonferenz Informatik*, 2018.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., and Moore, R. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment* vol. 202, pp. 18–27, 2017.

Guedes, T., Silva, V., Camata, J., Bedo, M. V., Mattoso, M., and de Oliveira, D. C. Towards an empirical evaluation of scientific data indexing and querying. *Journal of Information and Data Management* 9 (1): 84–84, 2018.

Guo, H., Liu, Z., Jiang, H., Wang, C., Liu, J., and Liang, D. Big earth data: a new challenge and opportunity for digital earth's development. *International Journal of Digital Earth* 10 (1): 1–12, 2017.

Hawkins, D. M. The problem of overfitting. *Journal of chemical information and computer sciences* 44 (1): 1–12, 2004.

Ji, C., Shao, Q., Sun, J., Liu, S., Pan, L., Wu, L., and Yang, C. Device data ingestion for industrial big data platforms with a case study. *Sensors* 16 (3): 279, 2016.

Kiran, M., Murphy, P., Monga, I., Dugan, J., and Baveja, S. S. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 2785–2792, 2015.

Lin, F.-C., Chung, L.-K., Wang, C.-J., Ku, W.-Y., and Chou, T.-Y. Storage and processing of massive remote sensing images using a novel cloud computing platform. *GIScience & Remote Sensing* 50 (3): 322–336, 2013.

Lin, J. The lambda and the kappa. *IEEE Internet Computing* 21 (05): 60–66, 2017.

Lu, M., Pebesma, E., Sanchez, A., and Verbesselt, J. Spatio-temporal change detection from multidimensional arrays: Detecting deforestation from modis time series. *ISPRS Journal of Photogrammetry and Remote Sensing* vol. 117, pp. 227–236, 2016.

Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., and Jie, W. Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems* vol. 51, pp. 47–60, 2015.

Nielsen, D. *Tree Boosting With XGBoost-Why Does XGBoost Win" Every" Machine Learning Competition?* M.S. thesis, NTNU, 2016.

Oliveira, S. S. T., Rodrigues, V., and Martins, W. S. Smart: Uso de aprendizado de máquina para filtragem e recuperação eficiente de dados espaciais e temporais em big data. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*. SBC, pp. 85–96, 2020.

Oliveira, S. S. T. d. *Explorando paralelismo em big data no processamento de séries temporais de imagens de sensoriamento remoto*. Ph.D. thesis, Universidade Federal de Goiás. Instituto de Informática, 2019.

Opitz, D. and Maclin, R. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research* vol. 11, pp. 169–198, 1999.

Patterson, J. Lumberyard: Time series indexing at scale. In *OSCON 2011 - O'Reilly Conferences*. Portland, USA, 2011.

Polikar, R. Ensemble based systems in decision making. *IEEE Circuits and systems magazine* 6 (3): 21–45, 2006.

Procopiuc, O., Agarwal, P. K., Arge, L., and Vitter, J. S. Bkd-tree: A dynamic scalable kd-tree. In *International Symposium on Spatial and Temporal Databases*. Springer, pp. 46–65, 2003.

Rajaraman, A. and Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2011.

Rathore, M. M. U., Paul, A., Ahmad, A., Chen, B.-W., Huang, B., and Ji, W. Real-time big data analytical architecture for remote sensing application. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 8 (10): 4610–4621, 2015.

Romani, L. A., Ávila, A. M. H., Zullo Jr, J., Traina Jr, C., and Traina, A. J. Mining relevant and extreme patterns on climate time series with clipsminer. *Journal of Information and Data Management* 1 (2): 245–245, 2010.

Shrivastava, S. A review of spatial big data platforms, opportunities, and challenges. *IETE Journal of Education* 61 (2): 80–89, 2020.

Song, W., Jin, B., Li, S., Wei, X., Li, D., and Hu, F. Building spatiotemporal cloud platform for supporting gis application. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* vol. 1, pp. 55–62, 2015.

Stonebraker, M., Brown, P., Poliakov, A., and Raman, S. The architecture of scidb. In *International Conference on Scientific and Statistical Database Management*. Springer, pp. 1–16, 2011.

Tahmassebpour, M. and Otaghvari, A. Increase efficiency big data in intelligent transportation system with using iot integration cloud. *Journal of Fundamental and Applied Sciences* 8 (3S): 2443–2461, 2016.

Thacker, U., Pandey, M., and Rautaray, S. S. Performance of elasticsearch in cloud environment with ngram and non-ngram indexing. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. IEEE, pp. 3624–3628, 2016.

Van Den Bergh, F., Wessels, K. J., Miteff, S., Van Zyl, T. L., Gazendam, A. D., and Bachoo, A. K. Hitempo: a platform for time-series analysis of remote-sensing satellite data in a high-performance computing environment. *International journal of remote sensing* 33 (15): 4720–4740, 2012.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, pp. 5, 2013.

Wagner, W., Fröhlich, J., Wotawa, G., Stowasser, R., Staudinger, M., Hoffmann, C., Walli, A., Federspiel, C., Aspetsberger, M., Atzberger, C., et al. Addressing grand challenges in earth observation science: The earth observation data centre for water resources monitoring. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 2 (7), 2014.

Wang, F., Li, M., Mei, Y., and Li, W. Time series data mining: A case study with big data analytics approach. *IEEE Access* vol. 8, pp. 14322–14328, 2020.

Wang, Y., Yuan, J., Chen, X., and Bao, J. Smart grid time series big data processing system. In *Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2015 IEEE*. IEEE, pp. 393–400, 2015.

Wolfert, S., Ge, L., Verdouw, C., and Bogaardt, M.-J. Big data in smart farming-a review. *Agricultural Systems* vol. 153, pp. 69–80, 2017.

Xu, M., Zhao, L., Yang, R., Yang, J., Sha, D., and Yang, C. Integrating memory-mapping and n-dimensional hash function for fast and efficient grid-based climate data query. *Annals of GIS*, 2020.

You, S., Zhang, J., and Gruenwald, L. Large-scale spatial join query processing in cloud. In *2015 31st IEEE international conference on data engineering workshops*. IEEE, pp. 34–41, 2015.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. Apache spark: a unified engine for big data processing. *Communications of the ACM* 59 (11): 56–65, 2016.

Zhang, X., Khanal, U., Zhao, X., and Ficklin, S. Understanding software platforms for in-memory scientific data analysis: A case study of the spark system. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, pp. 1135–1144, 2016.

Zhang, X., Khanal, U., Zhao, X., and Ficklin, S. Making sense of performance in in-memory computing frameworks for scientific data analysis: A case study of the spark system. *Journal of Parallel and Distributed Computing* vol. 120, pp. 369–382, 2018.

Zhou, Z.-H. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.