# A User-centric View of Distributed Spatial Data Management Systems

**João Pedro de Carvalho Castro**[1], **Anderson Chaves Carniel**[1], **Cristina Dutra de Aguiar Ciferri**[1]

[1]Department of Computer Science – University of São Paulo – Brazil

jp.carvalhocastro@usp.br, accarniel@gmail.com, cdac@icmc.usp.br

***Abstract.*** *Distributed spatial data management systems (DSDMSs) represent a new technology capable of managing huge volumes of spatial data using parallel and distributed frameworks. An increasing number of DSDMSs have been proposed in the literature, requiring a comparison among them. However, comparisons available in the literature only provide a system-centric view of DSDMSs, which is essentially based on performance evaluations. Thus, there is a lack of comparisons based on the user-centric view, which is aimed to help users to understand how the characteristics of DSDMSs are useful to meet the specific requirements of their spatial applications. In this paper, we fill this gap in the literature. We provide a user-centric comparison of Hadoop-GIS, SpatialHadoop, SpatialSpark, GeoSpark, SIMBA, LocationSpark, SparkGIS, and Elcano, using as a basis an extensive set of criteria related to the characteristics of spatial data handling and to the aspects inherent to distributed systems. Based on this comparison, we introduce a set of guidelines to help users to choose an appropriate DSDMS. We also describe a case study to illustrate the use of these guidelines.*

## 1. Introduction

The analysis of spatial data is a core issue for corporations that use geographic location to take strategic decisions and to enhance the user experience. These corporations have a massive advantage over their competitors and are able to react quickly to business conditions changes. Nowadays, the volume of spatial data is growing increasingly fast, mainly due to the wide variety of applications that harvest this data, such as mobile and Internet of Things applications. Therefore, there is a demand for new technologies capable of managing huge volumes of spatial data.

*Distributed spatial data management systems* (DSDMSs) have emerged as a solution to this demand. They provide specialized functionalities aimed to process and index huge volumes of vector spatial data using parallel and distributed frameworks, such as the Apache Hadoop MapReduce[1] and the Apache Spark[2] [García-García et al. 2017]. The Apache Hadoop MapReduce is based on a generic programming model composed of map and reduce functions, while the Apache Spark is based on in-memory computation and on a Resilient Distributed Dataset (RDD) abstraction. DSDMSs are developed on the top of these frameworks, inheriting their characteristics and advantages, and providing extended functionalities to deal with spatial data.

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

Several DSDMSs have been proposed in the literature, which are classified as Hadoop- or Spark-based systems. The most remarkable Hadoop-based systems are Hadoop-GIS [Aji et al. 2013] and SpatialHadoop [Eldawy and Mokbel 2015]. The state-of-the-art Spark-based systems include SpatialSpark [You et al. 2015], GeoSpark [Yu et al. 2015], SIMBA [Xie et al. 2016], LocationSpark [Tang et al. 2016], SparkGIS [Baig et al. 2017], and Elcano [Engélinus and Badard 2018]. However, each DSDMS has its own characteristics and introduces different functionalities to deal with spatial data in parallel and distributed environments. Hence, they integrate two different perspectives, the characteristics of spatial data handling [Güting 1994, OGC 2018] and the aspects inherent to distributed systems [Pandey et al. 2018].

Due to the variety of DSDMSs and provided functionalities, users who design, develop, and implement spatial applications for corporations face the challenge of choosing one system over the others. Indeed, the arguments behind choosing a given DSDMS depend on the purpose of the application. There are spatial applications that process ad-hoc spatial queries (e.g., [Wiemann et al. 2018]). For these applications, the chosen system should provide support for processing different types of spatial operations, such as topological predicates (e.g., contains, inside, and meet), geometric set operations (e.g., union, intersection), and numerical operations (e.g., area, distance). Other spatial applications require interoperability among different systems (e.g., [Lee and Reichardt 2005]); thus, the existence of different representations (e.g., textual and binary) for spatial objects (e.g., points, lines, and regions) is a requirement. Further, there are applications that require quick answers to specific spatial queries [Pandey et al. 2018]. In this case, the chosen system should provide indices specifically designed to answer these queries efficiently.

A review of related work aimed to compare DSDMSs shows that existing studies focus on comparing these systems experimentally (see Section 2). That is, they provide a *system-centric* view of DSDMSs based on performance evaluations. However, it is also important for users to understand how the characteristics of these systems are useful to meet the specific requirements of their spatial applications. To the best of our knowledge, there is no related work that provides a comparison based on this *user-centric* view.

The main goal of this work is to fill this gap in the literature by analyzing, from the *user-centric* point of view, the following DSDMSs: Hadoop-GIS, SpatialHadoop, SpatialSpark, GeoSpark, SIMBA, LocationSpark, SparkGIS, and Elcano. We introduce the contributions described as follows.

- Comparison of the DSDMSs using an extensive set of criteria related to the characteristics of spatial data handling and to the aspects inherent to distributed systems.
- Proposal of a set of guidelines, based on the comparison, to help users to identify the systems that most meet the specific requirements of their spatial applications.
- Description of a case study using GeoSpark to illustrate the use of the guidelines.

This paper is organized as follows. Section 2 reviews related work. Section 3 defines the set of criteria and compares the analyzed DSDMSs. Section 4 introduces our guidelines. Section 5 describes the case study. Finally, Section 6 concludes the paper.

## 2. Related Work

We survey related work considering two groups. The first one refers to approaches that introduce DSDMSs. Because the proposal of these systems depends on computational

advances such as those related to parallel and distributed frameworks for processing big data, the first DSDMSs available in the literature are Hadoop-based. Here, we are interested in Hadoop-GIS [Aji et al. 2013] and SpatialHadoop [Eldawy and Mokbel 2015]. The latest systems have been Spark-based, i.e., SpatialSpark [You et al. 2015], GeoSpark [Yu et al. 2015], SIMBA [Xie et al. 2016], LocationSpark [Tang et al. 2016], SparkGIS [Baig et al. 2017], and Elcano [Engélinus and Badard 2018]. Further, the work of García-García et al. (2017) introduces algorithms for optimizing distance join queries and implements them using SpatialHadoop and LocationSpark. Differently from our work, these approaches only briefly and technically summarize system by system. That is, they do not conduct a comparison among them considering the characteristics of spatial data handling and the aspects inherent to distributed systems. They also do not propose guidelines for users.

The second group refers to approaches aimed to provide a performance comparison among DSDMSs. In this context, there are only two related works that have been proposed in the literature. In Hagedorn et al. (2017), a performance evaluation is conducted focusing on the spatial filter and join operators for the following DSDMSs: Hadoop-GIS, SpatialHadoop, SpatialSpark, GeoSpark, and STARK[3], which is a spatial-temporal query processing extension that integrates into any Spark application. A broader performance evaluation is introduced in Pandey et al. (2018). First, the authors briefly survey SpatialHadoop, Hadoop-GIS, SpatialSpark, GeoSpark, Simba, Magellan[4], and LocationSpark. Then, they present extensive experiments involving the last five DSDMSs, considering five different spatial queries (i.e., range query, kNN query, spatial joins between distinct spatial data types, distance join, and kNN join) and four different data types (i.e., points, lines, rectangles, and polygons). However, Hagedorn et al. (2017) and Pandey et al. (2018) provide a *system-centric* view of DSDMSs, which is aimed to compare these systems based on their performance only. On the other hand, we compare DSDMSs considering the *user-centric* view, which is aimed to help users to understand how the characteristics of these systems are useful to meet the specific requirements of their spatial applications. Another differential of our work is that we also compare SparkGIS and Elcano, which are recently published DSDMSs.

In this paper, we analyze DSDMSs with publications in the literature and that support spatial data only. Thus, Magellan and STARK are not included in our comparisons. Further, carrying out performance evaluations of the DSDMSs is out of the scope of our paper due to the *user-centric* view. In this context, we consider the work of Pandey et al. (2018) as the state-of-the-art *system-centric* view. We use their findings in Section 3.2 and in Section 4 to complement our work.

## 3. User-Centric Comparative Analysis

In this section, we introduce a detailed *user-centric* comparison among the following DSDMSs: Hadoop-GIS, SpatialHadoop, SpatialSpark, GeoSpark, SIMBA, LocationSpark, SparkGIS, and Elcano. We use an extensive set of criteria that integrate two different perspectives: (i) the characteristics of spatial data handling (Section 3.1); and (ii) the aspects inherent to distributed systems (Section 3.2).

[3]https://github.com/dbis-ilm/stark

[4]https://github.com/harsha2010/magellan

**Table 1. A *User-centric* view of DSDMSs, considering the characteristics of spatial data handling.**

| DSDMS | Spatial Data Types | Representation of Spatial Objects | Geometric Set Operations | Topological Predicates | Numerical Operations | Spatial Indexing |
|---|---|---|---|---|---|---|
| Hadoop-GIS | ✓ | textual only | union and intersection only | ✓ | ✓ | ✓ |
| SpatialHadoop | simple points, rectangles, and polygons only | textual only | union only | limited | for queries only | ✓ |
| SpatialSpark | ✓ | textual only | ✓ | ✓ | ✓ | ✓ |
| GeoSpark | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SIMBA | simple points and rectangles only | textual only | no | limited | ✓ | ✓ |
| LocationSpark | simple points and rectangles only | user should implement | no | limited | for queries only | ✓ |
| SparkGIS | at least simple spatial data types | not specified | not specified | not specified | at least for queries | ✓ |
| Elcano | ✓ | textual only | ✓ | ✓ | ✓ | ✓ |

### 3.1. Support for Spatial Data Types and Their Operations

Table 1 compares the DSDMSs considering the following characteristics of spatial data handling [Güting 1994, OGC 2018]: spatial data types, representation of spatial objects, geometric set operations, topological predicates, numerical operations, and spatial indexing techniques. A *checkmark icon* in a cell indicates that the DSDMS completely fulfills the corresponding criterion. A *not specified* expression indicates that no information is provided in the research paper that introduces the DSDMS, and the criterion was not further analyzed because the implementation of the system is not available yet.

Before detailing the motivation and the context behind each aforementioned criterion, it is important to note that the underlying library used by the DSDMSs to handle spatial objects impacts directly on their capabilities regarding the management of spatial data. SpatialSpark, GeoSpark, and Elcano are based on the JTS library[5]. Hence, they fulfill almost all the criteria since this library follows the OGC specifications [OGC 2018].

**Spatial data types.** The support for spatial data types is fundamental to adequately represent geographical phenomena of different dimensions, such as points, lines, and regions (i.e., polygons) [Güting 1994]. Regions might also be specialized to other types, such as rectangles. Further, spatial data types can be simple or complex; simple spatial data types provide only single-component objects, while complex spatial data types provide versatile spatial objects with finitely many components. Hadoop-GIS, SpatialSpark, GeoSpark, and Elcano provide support for simple and complex spatial data types. The remaining DSDMSs limit their scope. For instance, SIMBA and LocationSpark provide support for simple points and rectangles only. SpatialHadoop, however, is an exception. Despite its

[5]`https://locationtech.github.io/jts/`

limited support for spatial data types, it can be extended, allowing users to define their own geometry data types. Regarding SparkGIS, it may provide features to manipulate complex spatial data types, but this cannot be affirmed since its implementation is not available yet.

**Representation of spatial objects.** Spatial objects can assume distinct representations that are used for different purposes, such as interoperability between applications, loading of spatial objects into DSDMSs, and visualization of spatial objects. The OGC standard specifies several representations [OGC 2018]. For instance, GML and GeoJSON are textual representations of spatial objects useful for visualizing objects in web-based applications. Another example is WKB, a binary representation that is useful to make fast data transferring between applications. Only GeoSpark supports textual and binary representations. Regarding LocationSpark, it does not provide encapsulated functions capable of processing any type of representation, burdening the user to implement these functions.

**Geometric set operations.** This type of spatial operation calculates the geometric intersection, geometric union, and geometric difference of two spatial objects [Güting 1994]. The result of these operations is another spatial object that can be used in further spatial analysis; thus, allowing users to take more flexible strategic decisions and enhancing the user experience. With the exception of the DSDMSs that employ the JTS library, geometric set operations are not fully supported by Hadoop-GIS and SpatialHadoop, which include up to two operations, or are not supported at all by the others systems.

**Topological predicates.** Spatial queries commonly required in spatial applications often make use of topological predicates [Gaede and Günther 1998], such as overlap, contains, and intersects. Examples of typical spatial queries include: (i) spatial selections that return a set of spatial objects satisfying a topological predicate given a search object, (ii) range queries that yield all spatial objects satisfying a topological predicate given a rectangular-shaped object called query window, and (iii) spatial joins that combine different sets of spatial objects according to a topological predicate. From the *user-centric* view, it is important to a DSDMS to offer native support for topological predicates because users can define the specific queries required by their applications. Hadoop-GIS, SpatialSpark, GeoSpark, and Elcano satisfy this requirement. On the other hand, SpatialHadoop, SIMBA, and LocationSpark do not allow the specification of ad-hoc spatial queries with topological predicates. Instead, they only support a subgroup of optimized spatial queries with a predetermined set of predicates. For instance, range and spatial join queries in SpatialHadoop can only be executed with the predicate overlap.

**Numerical operations.** This type of operation returns numbers calculated from geometric properties of spatial objects [Güting 1994], such as the length of lines and the area of regions. Further, numerical operations can be employed to execute distance-based spatial queries, such as the $k$-nearest neighbor query that returns a set of $k$ spatial objects nearest to an origin location. For instance, distance-based spatial queries are useful in spatial neighbourhood analysis. All DSDMSs provide at least some support for numerical operations. In SpatialHadoop, LocationSpark, and SparkGIS, the numerical operations are performed only within encapsulated functions that execute spatial queries. However, SparkGIS may provide other features to manipulate numerical operations, but this cannot be affirmed since its implementation is not available yet. Finally, regarding numerical op-

**Table 2. A *user-centric* view of DSDMSs, considering the aspects inherent to distributed systems.**

| DSDMS | Underlying Technology | Doc. Completeness | Spatial Partitioning | Distributed Indexing | Query Language | Visualization Module |
|---|---|---|---|---|---|---|
| Hadoop-GIS | Hadoop-based | ✓ | ✓ | ✓ | HiveSQL-based | limited |
| SpatialHadoop | Hadoop-based | ✓ | ✓ | ✓ | Pigeon-based | for simple spatial data types only |
| SpatialSpark | Spark-based | limited | ✓ | local indices only | no | no |
| GeoSpark | Spark-based | ✓ | ✓ | local indices only | SQL-based | for simple spatial data types only |
| SIMBA | Spark-based | limited | ✓ | ✓ | SQL-based | no |
| LocationSpark | Spark-based | limited | ✓ | ✓ | no | no |
| SparkGIS | Spark-based | unavailable | ✓ | ✓ | no | limited |
| Elcano | Spark-based | unavailable | not specified | not specified | SQL-based | no |

erations that extract geometric characteristics, only SpatialSpark, GeoSpark, and Elcano provide this type of support since they are based on the JTS library.

**Spatial indexing techniques.** The use of a spatial index is one of the most common techniques employed to accelerate spatial query processing [Gaede and Günther 1998]. Many different spatial indices have been proposed in the literature, such as the R-trees and the Quadtrees. In general, DSDMSs employ spatial indices for two main purposes: (i) to process spatial queries in slave nodes; and (ii) to distribute data among slave nodes and possibly reduce the number of partitions visited during a spatial query (Section 3.2). Because of these advantages, spatial indices are supported by all compared DSDMSs.

### 3.2. Support for Aspects Inherent to Distributed Systems

Table 2 compares the DSDMSs considering the following aspects inherent to distributed systems: underlying technology, documentation completeness, spatial partitioning, distributed indexing, and query language. A *checkmark icon* is employed if a DSDMS completely fulfills a corresponding criterion. A *not specified* expression indicates that no information is provided in the research paper that introduces the DSDMS, and the criterion was not further analyzed because the implementation of the system is not available yet. We discuss the motivation and the context behind each criterion as follows.

**Underlying technology.** The underlying technology used to implement a DSDMS impacts on the performance of spatial applications because of the I/O cost. Hadoop-based systems need to write intermediate data to disk, while Spark-based systems store intermediate data in the main memory through the use of RDDs. Thus, according to Pandey et al. (2018), Spark-based systems usually deliver better performance.

**Documentation completeness.** A complete, accurate, and up-to-date documentation is a required prerequisite to help users who design, develop, and implement spatial applications, especially when these users are dealing with state-of-the-art systems. The lack of documentation may impact negatively in the application development, requiring extra

time to develop the application and burdening the users. Hadoop-GIS, SpatialHadoop, and GeoSpark stand out since these systems provide a dedicated website with expressive content, such as list of features, detailing of installation procedures, and several tutorials describing how to execute spatial operations. The available documentations of SpatialSpark, SIMBA, and LocationSpark provide only a short description and a quick example of some of their operations. SparkGIS and Elcano are exceptions. They do not have a public version released yet, and therefore do not provide documentation.

**Spatial partitioning.** The partitioning of data across cluster nodes is a technique frequently used in parallel and distributed computing. This technique usually speed up query processing by taking advantage of data locality. By using spatial partitioning techniques, a DSDMS is able to use the spatial characteristics of the dataset as the criterion for the partitioning, improving the performance of spatial queries. Several DSDMSs employ spatial partitioning techniques, with Hadoop-GIS, SpatialHadoop, SIMBA, and SparkGIS providing the most expressive quantity of algorithms available.

**Distributed indexing.** The concept of spatial indexing in a parallel and distributed environment is strongly related to spatial partitioning. Commonly, two data structures are created: (i) a global index, located on the master node, pointing to each data partition, and (ii) multiple local indices, each located on a data partition, pointing to the data inside the partition. The global index, which is created with the same data structure used for spatial partitioning, is applied before executing a spatial query in order to prune unnecessary partitions. This global index, however, is not employed by SpatialSpark and GeoSpark, which include only local indices. On the other hand, global and local indices are employed by Hadoop-GIS, SpatialHadoop, SIMBA, LocationSpark, and SparkGIS, introducing several benefits as discussed in Section 3.1.

**Query language.** Extending existing query languages is an essential characteristic that a DSDMS should provide to simplify the manipulation of spatial objects. DSDMSs that extend well-known query languages, such as SQL, usually reduce the learning curve of users. That is, they enable users to quickly familiarize themselves with the features of the system. GeoSpark, SIMBA, and Elcano distinguish themselves because they extend SparkSQL to support the execution of spatial queries. Hadoop-GIS and SpatialHadoop also extend existing query languages, but those are based on other standards (HiveSQL and Pigeon, respectively). The remainder DSDMSs do not offer query language, requiring extra efforts from users to retrieve and manage spatial objects.

**Visualization module.** Providing a module for visualizing results of spatial queries is an important aspect that enhances the user experience in spatial applications. For instance, visualizing the output of a spatial query in a map instead of in a text file enables users to visible interpret its content and to take quicker decisions over the identified problems. To provide a precise and complete map visualization of spatial objects, the generation of high-resolution maps is needed. Currently, only SpatialHadoop and GeoSpark fully support this feature. However, this support is restricted to simple spatial data types. On the other hand, Hadoop-GIS only provides a simple tool that allows users to visualize the boundaries of partitions. Regarding SparkGIS, its support for visualizing spatial objects depends on the implementation of plugins that extend its functionalities.

## 4. User-centric Guidelines

In this section, we propose a set of *user-centric* guidelines to help users to identify the most appropriate DSDMSs to their needs, according to the analyses described in Section 3. Because the context and objectives behind the development of applications are very variable, we provide a small but general characterization that can be later specialized based on the requirements of each application. Thus, we propose guidelines defined on the *focus of spatial applications*. Each guideline also lists the DSDMSs that meet its specification. Non-listed DSDMSs may offer some related functionalities, but should require extra implementation efforts from users.

**Guideline 1. Focus on executing ad-hoc spatial queries**. This guideline is based on spatial applications that need to process spatial queries without specific formats. An example is an application for analyzing heterogeneous and distributed spatial data for environmental monitoring [Wiemann et al. 2018]. To fulfill Guideline 1, a DSDMS should provide support for an expressive variety of spatial operations, such as geometric set operations, topological predicates, and numerical operations. Based on our analyses, the following DSDMSs fulfill Guideline 1: SpatialSpark, GeoSpark, and Elcano.

**Guideline 2. Focus on the interoperability among different systems.** This guideline is based on spatial applications that need to communicate with each other, such as those that integrate heterogeneous spatial data from different sources. For instance, the integration of public and private urban transportation spatial data [Smarzaro et al. 2017]. To fulfill Guideline 2, a DSDMS should provide support for all spatial data types since the representation of spatial phenomena can be different in the sources. Further, spatial objects should be exchangeable by using textual or binary representations. Based on our analyses, GeoSpark fulfills Guideline 2. Hadoop-GIS, SpatialSpark, and Elcano partially fulfill this guideline because they provide support for textual representations only.

**Guideline 3. Focus on characteristics based on well-known standards.** This guideline is based on spatial applications that require the use of well-known and accepted concepts, such as terms and expressions employed in the literature, query languages, and representations of spatial objects. An example is the application of some open standards for homeland security networks [Lee and Reichardt 2005]. Another example is the web-based application detailed in Wiemann et al. (2018). To fulfill Guideline 3, a DSDMS should employ well-known and accepted concepts in its design. It also should follow Guideline 2. Based on our analyses, GeoSpark fulfills Guideline 3. SpatialSpark, SIMBA, and Elcano also consider several aspects of this guideline, but introduce limitations related to the lack of a binary representation of spatial objects based on well-known standards.

**Guideline 4. Focus on spatial data visualization.** Spatial applications usually require the use of graphical user interfaces through which users are able to manipulate spatial objects, share findings by plotting spatial objects in maps, and enrich their decision-making. For instance, applications analyzing traffic data require the visualization of spatial data to better understand transportation systems [Chen et al. 2015]. To fulfill Guideline 4, a DSDMS should provide visualization modules without restricting the type of spatial data being manipulated. To the best of our knowledge, there is no compared DSDMS that completely fulfills this guideline. Hadoop-GIS, SpatialHadoop, GeoSpark, and SparkGIS only provide a limited support for visualizing spatial objects.

**Guideline 5. Focus on efficiently processing spatial queries.** Reducing the time spent to process spatial queries is a common requirement of spatial applications. For instance, García-García et al. (2017) propose algorithms for optimizing distance joins queries. To fulfill Guideline 5, a DSDMS should include optimized- and specialized-algorithms for processing spatial queries, including the use of indices. Here, the findings about the performance evaluation of DSDMSs described in Pandey et al. (2018) should be used as foundation. Pandey et al. (2018) also provide the best parameters of these systems to process several types of spatial queries.

## 5. Case Study

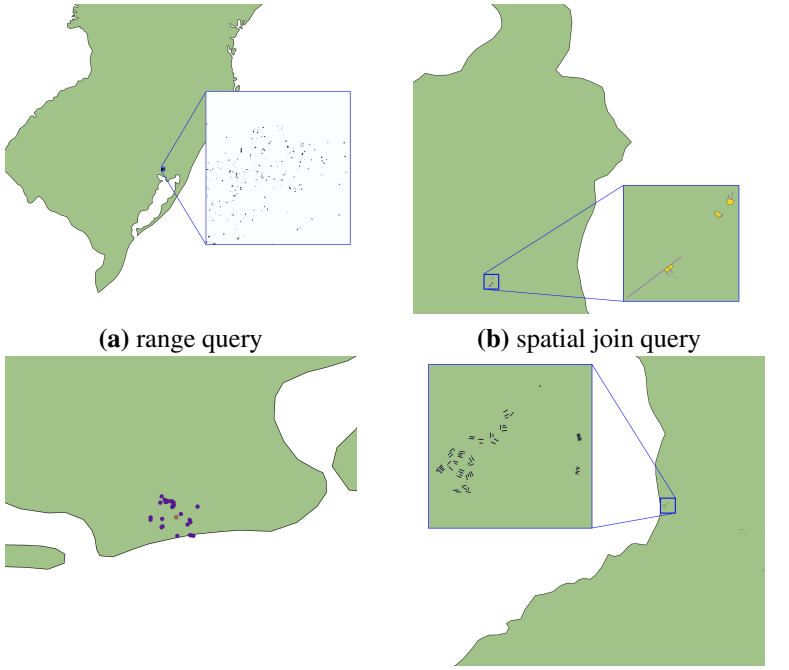
In this section, we describe a case study that illustrates the use of the proposed guidelines. We use a spatial application containing real spatial objects extracted from OpenStreetMaps that correspond to buildings, highways, and single locations of Brazil [Carniel et al. 2017]. They are represented by regions, lines, and points respectively. This application handles these objects in spatial queries that analyze the infrastructure situation of different places, such as farms, schools, and roads. Users who design, develop, and implement this spatial application should consider the following requirements to decide which DSDMS to choose. The application should manage spatial objects represented by simple and complex data types. The spatial objects are stored in CSV files. Each line of these files has the format $(geo, desc)$, where *geo* is the WKT representation of the spatial object and *desc* is its description. The application should also support ad-hoc spatial queries possibly containing geometric set operations, topological predicates, and numerical operations. Further, the application should provide good performance results. Finally, it is important to note that users have some previous knowledge of SQL.

The application's requirements indicate that users should take into account Guidelines 1, 3, and 5 (Section 4). As a result, users choose GeoSpark as the most appropriate DSDMS since it fulfills the requirements. Regarding Guideline 5, users should apply the same values of parameters as described in Pandey et al. (2018) to guarantee the best elapsed times for spatial queries. These parameters are: (i) the R-tree as the local index, and (ii) the Quadtree as the spatial partitioning technique.

**Data loading.** First, spatial objects from the CSV files are loaded into GeoSpark by using GeoSparkSQL. Because this module is an extension of SparkSQL, the files are loaded into a structure called DataFrame, which resembles a relational table. Next, the column that stores the textual representation of the spatial data should be transformed into a geometry column. GeoSparkSQL provides a function capable of transforming a WKT representation into a spatial object, which can be manipulated in SQL queries (Guideline 4). The following query is an example of how the column that contains the WKT representation of the regions stored in $brazil\_buildings$ can be transformed into a geometry column:

```
SELECT ST_GeomFromWKT(brazil_buildings.geo) AS geo,
       brazil_buildings.desc AS desc
FROM brazil_buildings
```

After loading all spatial objects, users can execute ad-hoc spatial queries on the DataFrame (Guideline 1). We define four ad-hoc spatial queries as samples for our application. These queries are described as follows.



**(a)** range query **(b)** spatial join query

**(c)** k-nearest neighbours query **(d)** geometric set manipulation query

**Figure 1. Visualization of the query results.**

**Range query.** The first query returns all schools located inside a query window (QW), which is a rectangle that corresponds to 0.001% of the total extent of Brazil. The results of this query enable users to analyze the school coverage in a given range. The command that express this query employs the topological predicate *contains* as follows:

```
SELECT brazil_buildings.geo
FROM brazil_buildings
WHERE ST_Contains(QW, brazil_buildings.geo)
      AND brazil_buildings.desc = 'school'
```

**Spatial join query.** The next query returns all tracks (i.e., rough road used by agricultural or similar vehicles) that intersect a building. This query is useful to analyze if tracks should be improved or not. The command that express this query employs the topological predicate *intersects* as follows:

```
SELECT brazil_buildings.geo, brazil_highways.geo
FROM brazil_buildings, brazil_highways
WHERE ST_Intersects(brazil_buildings.geo, brazil_highways.geo)
      AND brazil_highways.desc = 'track'
```

**K-nearest neighbours query.** The next query yields the 30 nearest energy towers from the Olympic Arena of Rio de Janeiro (PT) to analyze the infrastructure situation of an important neighborhood. To this end, the following distance-based query can be written:

```
SELECT brazil_points.geo, ST_Distance(brazil_points.geo, PT) as d
FROM brazil_points
```

```
WHERE brazil_points.desc = 'tower'
ORDER BY d
LIMIT 30
```

**Geometric set manipulation query.** The final query returns the total area of all farms in Brazil. Hence, we need to compute the geometric union among all farms and then calculate its area, as follows:

```
SELECT ST_Area(ST_Union_Aggr(brazil_buildings.geo))
FROM brazil_buildings
WHERE brazil_buildings.desc = 'farm'
```

Figure 1 depicts the spatial objects returned by these queries. Zooming was applied to better visualize portions of the result; thus, the result was not completely displayed for some queries. We use QGIS[6] to show the queries' results because the visualization module GeoSpark-Viz does not allow the visualization of complex spatial objects.

## 6. Conclusions and Future Work

In this paper, we provide a comparative analysis of the following up-to-date DSDMSs: Hadoop-GIS, SpatialHadoop, SpatialSpark, GeoSpark, SIMBA, Location-Spark, SparkGIS, and Elcano. Because the analysis is performed from the *user-centric* view, it is aimed to help users to understand how the characteristics of DSDMSs are useful to meet the specific requirements of their spatial applications. Based on the comparisons, we propose a set of guidelines to help users to choose an appropriate DSDMS to design, develop, and implement their spatial applications. Finally, we describe a case study using GeoSpark to illustrate the use of the guidelines. Future work includes the *user-centric* comparison of other DSDMSs such as Magellan and STARK. Another future work is to describe the case study using each surveyed DSDMS.

## Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This work has also been supported by CNPq and by the São Paulo Research Foundation (FAPESP). Anderson C. Carniel has been supported by the grant #2015/26687-8, FAPESP. Cristina D. A. Ciferri has been supported by the grant #2018/22277-8, FAPESP.

## References

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. H. (2013). Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. *VLDB Endowment*, 6(11):1009–1020.

Baig, F., Vo, H., Kurç, T. M., Saltz, J. H., and Wang, F. (2017). SparkGIS: Resource aware efficient in-memory spatial query processing. In *ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 28:1–28:10.

Carniel, A. C., Ciferri, R. R., and Ciferri, C. D. A. (2017). Spatial datasets for conducting experimental evaluations of spatial indices. In *Satellite Events of the Brazilian Symposium on Databases - Dataset Showcase Workshop*, pages 286–295.

[6] http://qgis.osgeo.org

Chen, W., Guo, F., and Wang, F. (2015). A survey of traffic data visualization. *IEEE Trans. on Intelligent Transportation Systems*, 16(6):2970–2984.

Eldawy, A. and Mokbel, M. F. (2015). SpatialHadoop: A MapReduce framework for spatial data. In *Int. Conf. on Data Engineering*, pages 1352–1363.

Engélinus, J. and Badard, T. (2018). Elcano: A geospatial big data processing system based on SparkSQL. In *Int. Conf. on Geographical Information Systems Theory, Applications and Management*, pages 119–128.

Gaede, V. and Günther, O. (1998). Multidimensional access methods. 30(2):170–231.

García-García, F., Corral, A., Iribarne, L., Mavrommatis, G., and Vassilakopoulos, M. (2017). A comparison of distributed spatial data management systems for processing distance join queries. In *European Conf. on Advances in Databases and Information Systems*, pages 214–228.

Güting, R. H. (1994). An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399.

Hagedorn, S., Götze, P., and Sattler, K. (2017). Big spatial data processing frameworks: Feature and performance evaluation. In *Int. Conf. on Extending Database Technology*, pages 490–493.

Lee, K. B. and Reichardt, M. E. (2005). Open standards for homeland security sensor networks. *IEEE Instrumentation Measurement Magazine*, 8(5):14–21.

OGC (2018). OpenGIS® Implementation Standard for Geographic Information - Simple Feature Access - Part 1: Common Architecture. Open Geospatial Consortium. Available at: `http://www.opengeospatial.org/standards/sfa`.

Pandey, V., Kipf, A., Neumann, T., and Kemper, A. (2018). How good are modern spatial analytics systems? *VLDB Endowment*, 11(11):1661–1673.

Smarzaro, R., Lima, T. F. M., and Davis, Jr., C. A. (2017). Could data from location-based social networks be used to support urban planning? In *Int. Conf. on World Wide Web Companion*, pages 1463–1468.

Tang, M., Yu, Y., Malluhi, Q. M., Ouzzani, M., and Aref, W. G. (2016). LocationSpark: A distributed in-memory data management system for big spatial data. *VLDB Endowment*, 9(13):1565–1568.

Wiemann, S., Karrasch, P., and Bernard, L. (2018). Ad-hoc combination and analysis of heterogeneous and distributed spatial data for environmental monitoring - design and prototype of a web-based solution. *International Journal Digital Earth*, 11(1):79–94.

Xie, D., Li, F., Yao, B., Li, G., Zhou, L., and Guo, M. (2016). Simba: Efficient in-memory spatial analytics. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 1071–1085.

You, S., Zhang, J., and Gruenwald, L. (2015). Large-scale spatial join query processing in cloud. In *Int. Conf. on Data Engineering Workshops*, pages 34–41.

Yu, J., Wu, J., and Sarwat, M. (2015). GeoSpark: a cluster computing framework for processing large-scale spatial data. In *ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 70:1–70:4.