Elcano: A Geospatial Big Data Processing System based on SparkSQL

Jonathan Engélinus and Thierry Badard
Centre for Research in Geomatics (CRG), Laval University, Québec, Canada

Keywords: Elcano, ISO-19125, Magellan, Spatial Spark, GeoSpark, Geomesa, Simba, Spark SQL, Big Data.

Abstract: Big data are in the midst of many scientific and economic issues. Furthermore, their volume is continuously increasing. As a result, the need for management and processing solutions has become critical. Unfortunately, while most of these data have a spatial component, almost none of the current systems are able to manage it. For example, while Spark may be the most efficient environment for managing Big data, it is only used by five spatial data management systems. None of these solutions fully complies with ISO standards and OGC specifications in terms of spatial processing, and many of them are neither efficient enough nor extensible. The authors seek a way to overcome these limitations. Therefore, after a detailed study of the limitations of the existing systems, they define a system in greater accordance with the ISO-19125 standard. The proposed solution, Elcano, is an extension of Spark complying with this standard and allowing the SQL querying of spatial data. Finally, the tests demonstrate that the resulting system surpasses the current available solutions on the market.

1 INTRODUCTION

Today, it becomes crucial to develop systems able to manage efficiently huge amounts of spatial data. Indeed, the convergence of the Internet and cartography has brought forth a new paradigm called "neogeography". This new paradigm is characterized by the interactivity of location based contents and the possibility for the user to generate them (Mericksay and Roche, 2010). This phenomenon, in conjunction with the arrival in the market of new captors like the GPS chips in smartphones, resulted in the inflation of production and retrieval of spatial data (Badard, 2014). This new interest for cartography makes the process more complex as it becomes more and more difficult to manage and represent such large quantities of data by use of conventional tools (Evans et al, 2014).

The Hadoop environment (White, 2012), currently one of the most important projects of the Apache Foundation, is a *de facto* standard for the processing and management of Big data. This very popular tool, involved in the success of many start-ups (Fermigier, 2011), implements MapReduce (Dean and Ghemawat, 2008), an algorithm that allows the distribution of data processing among the servers of a cluster for a faster execution. The data to process are also distributed among the servers by the Hadoop Distributed File System (HDFS), which is provided by default with Hadoop. The result is a high degree of horizontal scalability, which can be defined as the ability to linearly increase the performances of a multi-server system to meet the user's requirements in terms of processing time. A real ecosystem of interoperable elements has been built up around Hadoop, which enables the management of such various aspects as streaming (e.g. Storm), serialization (e.g. Avro) and data analysis (e.g. Hive).

In 2014, the University of Berkeley's AMPLab started to develop a new element of the Hadoop ecosystem, which has since been taken over by the Apache Foundation, namely Spark (http://spark.apache.org/), which offers an interesting alternative to HDFS and MapReduce. In Spark, data and processing codes are distributed together in small blocks called RDD ("Resilient Distributed Dataset") on the whole cluster RAM. This architectural choice, which strongly limits hard drive accesses, makes Spark up to ten times faster than conventional Hadoop use, in some cases (Zaharia et al, 2010), although at the cost of a greater RAM load (Gu and Li, 2013). Furthermore, a

Engélinus, J. and Badard, T.
Elcano: A Geospatial Big Data Processing System based on SparkSQL.
DOI: 10.5220/0006794601190128
In *Proceedings of the 4th International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM 2018)*, pages 119-128
ISBN: 978-989-758-294-3

part of Spark called Spark SQL (Armbrust et al, 2015) dress up Spark RDD with a supplementary level called "DataFrames" which allows to organize received data from Spark into temporary tables and to query them with SQL language. Spark SQL minimizes as well the duration of Spark processes, thanks to the strategic optimization of queries and the serialization of data. Finally, it allows the definition of personalized data types (UDT: "User Defined Type") and personalized functions (UDF: "User Defined Function"), which permit respectively to get new kinds of data and processing available from SQL.

This opportunity to query Big data thanks to SQL is of paramount importance as it helps in their analysis, with the goal of a better understanding of the phenomena they represent on the ground. It also empowers analysts with new analytical capabilities, using a query language they already master on the day to day. Together with the availability of a growing amount of geospatial data, it is profitable to use these capabilities to analyze the spatial component of this huge amount of information, which, according to Franklin, is present in 80% of all business data (Franklin and Hane, 1992). According to a frequently mentioned research of the McKinsey cabinet (Manyika et al, 2011), a better use of Big data spatial localization could grant 100 billion USD to services providers and in the range of 700 billion USD to final users. Lastly, spatial Big data management finds itself in the midst of many important economical, scientific and societal issues. In this respect, Spark appears again as a promising solution, because it processes the spatial data at least more than 7 times faster than Impala, another Hadoop element managing the SQL (You, et al, 2015).

Today, some systems relying on Hadoop enable the management of massive spatial data, such as Hadoop GIS (Aji et al, 2013), Geomesa (Hugues et al, 2015) and Pigeon (Eldawy and Mokbel, 2014). But they are mainly about prototypes than mature technologies (Badard, 2014). In addition, most of them only relies on the core version of Hadoop without fully scaling the processing power of the RAM like it is achieved by Spark. For example, Spatial Hadoop (Eldawy and Mokbel, 2013) only uses the Map Reduce algorithm of Hadoop.

Among these systems, only five propose a management of spatial data relying on Spark. The first two systems, Spatial Spark (You, et al, 2015) and GeoSpark (Yu et al, 2015) only add a management of the spatial component to the basic version of Spark, which do not fully take advantage of all the capabilities (e.g. SQL querying) and performance of Spark. Hence, the current Spatial Spark version can only interact with data in command line mode instead of managing SQL queries. GeoSpark only uses its own spatial extension of the Spark RDD type, which does not directly comply with Spark SQL (Yu, 2017). The third, Magellan (Ram, 2015), defines spatial data types directly available in Spark SQL, but without correctly managing some spatial operations like the union of disjointed polygons, the symmetric differences involving more than a geometry and the creation of an envelope. The fourth, Simba (Xie et al, 2016), enables the querying of data in SQL for points only and without the possibility to trigger standard spatial functions. At last, the fifth prototype is the Geomesa extension, which can be used from Spark. The system is anyway limited in the spatial operations that it offers because it has been natively designed only for the research of points included in an envelope. Furthermore, it presents limited performances (Xie et al, 2016) in comparison with other solutions. That apparently could be explained by the fact that it imposes the use of a key store technology (Accumulo, https://accumulo.apache.org/) to store the spatial data to process.

As a conclusion, there is presently no system for the management of geospatial data that fully manages all kinds of 2D geometry data types and that enables their efficient and actionable SQL querying. Each model implemented in the five studied prototypes which pursue a similar goal presents limited capacities both on the types of geometry they support as well as on the spatial processing capabilities they offer. Details about this last point are given in the next section.

2 LIMITS IN THE GEOSPATIAL CAPABILITIES SUPPORTED BY CURRENT SOLUTIONS

In order to assess the capabilities of the different geospatial Big data management systems currently relying on Spark to fully manage the 2D spatial component, the ISO-19125 standard can profitably be used as a guideline. Indeed, the two parts of this standard respectively describe the 2D geometry types and the geospatial functions and operators (ISO 19125-1, 2004) and their expression in the SQL language (ISO 19125-2, 2004) that a system must implement to basically store 2D geospatial data and support its querying and its analysis in an

interoperable way. In this context, we will first introduce the geometry types supported by the different systems. Then we will analyze which spatial functions they give access to and whether they can be extended to easily implement the missing ones. Finally, we will study how they manage the spatial indexation issue, which is crucial when dealing with geospatial data.

2.1 Geometry Data Types

A system complying with ISO 19125-1 is supposed to handle the seven main 2D geometry types that can be built by linear interpolation. These can be divided into three simple types (point, polyline and polygon) and into four composite types (multipoint, multipolyline, multipolygon and geometry collection). Here is a study of how the current systems meet this standard.

Spatial Spark and GeoSpark integrate all these types of geometries because their model relies on the use of the JTS ("Java Topology Suite", https://www.locationtech.org/proposals/jts-topology-suite) library, which has been designed to meet ISO standards and OGC recommendations (Davis and Aquino, 2003). Geomesa also manages all the geometries in its current version (Commonwealth Computer Research, 2017), while Simba only manages the point.

The case of the HortonWorks Magellan system is more mixed. It enables the processing of points, polylines and polygons. This may seem sufficient if one assumes, as one of the designers of the system (Sriharasha, 2016) does, that compound geometries are reducible to tables of geometries. But in reality, such an approach can only lead to a dysfunctional system. Indeed, by not being able to explicitly create actual complex geometry, such arrays are not allowed as operands of a spatial function and their return as a result of a spatial operation like the union of disjoint polygons causes a type error.

In addition to its development, Magellan's limitations are also due to the use of ESRI Tools as a spatial library. The latter does not make it possible to process all the 2D geometry types defined by the ISO-19125 standard. It lacks the geometry collection type, while the multi-polygon type is only partially implemented. Furthermore, the adaptation of WKT ("Well-Known Text") provided by ESRI Tools does not comply with the ISO standards and the OGC recommendations.

The limitations of the different solutions studied in relation to the requirements of ISO-19125-1 are summarized in Table 1. Those related to ESRI Tools have been added to give an idea of the limits that they involve on the evolution of Magellan.

Table 1: Coverage of the different 2D geometry types specified by ISO-19125 in studied prototypes.

	Geo Spark	Spatial Spark	Simba	Geomssa	Magellan	ESRI
Point	Yes	Yes	Yes	Yes	Yes	Yes
Polyline	Yes	Yes	No	Yes	Yes	Yes
Polygon	Yes	Yes	No	Yes	Yes	Yes
Multi-Point	Yes	Yes	No	Yes	No	Yes
Multi-polyline	Yes	No	Yes	No	Yes	Yes
Multi-polygon	Yes	Yes	No	Yes	No	In part
Collection	Yes	Yes	No	Yes	No	No

2.2 Spatial Functions and Operators

(ISO 19125-2, 2004) specifies how the spatial functions (relations, operations, metric functions and methods), a spatial data management system should implement in SQL to comply with the ISO 19125-1 standard. It does not specify the way these methods have to be implemented. It only defines their signatures. These functions define the minimal set of operations a system must implement to enable basic and advanced spatial analysis capabilities. Even if these functions have been defined for querying data in classic spatial DBMS, their usage in geospatial Big data management systems still pertain. Nevertheless, the application of the ISO-19125-2 standard requires a system allowing SQL queries and personalized SQL functions. This section details how the five studied systems partly implement the standard and describes their extension capabilities.

Spatial Spark only uses the core of Spark. Indeed, it allows to work with RDD's but not with DataFrames or SQL queries. In this context, the application of the ISO 19125-2 standard to Spatial Spark seems impossible without a full reimplementation.

As we saw, GeoSpark extends the RDD type of Spark, and is therefore not directly compatible with Spark SQL. Nevertheless, one of its developers indicates that the integration of this point is planned for a future version of the system and that there would be an indirect way of changing these RDD's in DataFrames (Yu, 2017). But neither does he describe a general process for it, nor how to apply SQL queries afterwards. Indeed, the current version of GeoSpark does not seem to be compliant with the ISO 19125-2 standard because all geometry types cannot be managed from SQL queries.

Simba released its own adaptation of Spark SQL, which might enable the use of SQL queries and the

creation of User Defined Functions. In practice however, the only accessible geometry is the point. Furthermore, the syntactic analyzer does not always work properly. By example, it forces to write "IN" before "POINT(x, y)" even without a context of inclusion. Simba is therefore not a mature and reliable solution that could meet the ISO 19125-2 standard.

Until recently, Geomesa's Spark extension only used Spark's core. But a recent version tries to integrate Spark SQL. However, this solution remains restrained by the mandatory use of the CQL format and the Accumulo database (Commonwealth Computer Research, 2017). Indeed, Geomesa does not allow an autonomic and agnostic implementation of ISO-19125-2.

Magellan does not directly manage SQL either. But it defines User Data Types for the point. It is therefore tempting to assume that the addition of Used Defined Functions to its model should be enough to allow the SQL functions of the ISO-19215-2 standard. In practice however, the extension of Magellan with these functions only covers two thirds of spatial relations, half of the spatial operations and a small part of spatial methods specified by the ISO-19125-2 standard. These limitations are due to both implementation errors and the choice of the ESRI Tools library, which only partially meets the ISO-19125-2 standard.

In their current states therefore, none of the studied systems totally comply with the ISO-19125 standard.

2.3 Spatial Indexation Management

Spatial indexation can be defined as the reorganization of spatial data, typically by using their proximity relations, with the purpose of accelerating their processing (Eldawy and Mokbel, 2015). Four of the studied systems provide a spatial indexation component, but which is never both efficient and extensible.

The spatial indexation component of Spatial Spark uses directly the methods of the JTS spatial library, which is not conceived for Big data processing in a multi-server environment. GeoSpark proposes a more integrated and efficient spatial indexation module (Yu, 2017), but without the possibility of managing it with SQL queries. The indexation component of Simba is described as more efficient by its developers (Xie et al, 2016), but has important limitations and bugs we already covered. Finally, Geomesa offers poor performances because it relies on a specific database system (Xie et al, 2016), which drastically increases the processing time.

2.4 Synthesis of Limitations

Table 2 sums up the main limitations of the studied systems. It first recalls their most problematic limitations. Then it reminds the geometry types they support and as a result their degree of conformance to the ISO 19125 standard. Next, it indicates whether they manage SQL and whether they comply with the ISO-19125-2 standard.

Table 2: Limitations of current spatial Big data processing systems.

	Magellan	**Spatial Spark GeoSpark**	**Simba**	**Geomesa**
Main limitation	Use a limited spatial library	Inextensible to SQL	Syntactic bugs, no extensible	Force to use a NoSQL database
Types of geometries	Only simples	All	Only point	All
ISO-19125-1	In part	Yes	In part	Yes
SQL management	No, but extensible	No	Yes (replace Spark SQL)	Yes, limited by CQL
ISO-19125-2	In part (by extension)	No	No	In part
Spatial indexation	No		Yes, but not efficient or not extensible	

Next section presents a new system designed for the efficient and interoperable management and rapid processing of geospatial Big data (vector data only). It relies on Spark and overcomes identified limitations present in current state-of-the-art solutions. This prototype is named Elcano. Its release as an open source project has not yet been performed but it is envisaged.

3 PRESENTATION OF ELCANO

The main objective leading the design of Elcano is to model a spatial Big data processing and management system that surpasses the other systems studied here. It must then integrate each 2D geometry types defined in the ISO-19125 standard. It must also enable the use of associated spatial functions, in order to improve the analysis of spatial phenomena. All spatial relations, operations and methods defined by the ISO-19125-2 standard must then be implemented by Elcano. For example, a call to the SQL function ST_Intersects has to indicate if two generic geometry objects intersect or not. The

system must also allow to load spatial data in a simple and generic way, for it to be easy to feed and to extend toward other formats. It also must ensure data persistence in memory, in a compact manner so that it enables a faster processing of the geospatial component. It has to be easily extensible in order to potentially support new geometry types or extensions to the geometry types defined in the ISO-19125 standard (for example the inclusion of elevation in geometric features definition, i.e. 2.5D data). Finally, it must offer good processing performances in comparison with current processing systems. A model seeking to meet these objectives is presented and justified below.

3.1 Architecture

Figure 1 illustrates the model on which Elcano is based. In this model, classes of the geometry package integrate the elementary geometries and spatial functions linked to Elcano.

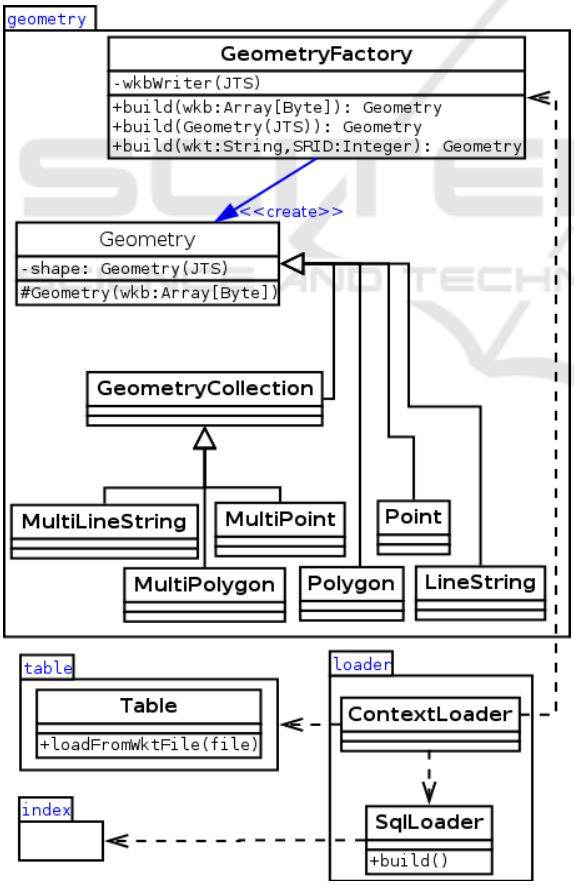

Figure 1: Elcano's model.

The loader package enables the use of SQL spatial functions. Data persistence for processing and data retrieval is managed by the "Table" class, together with the support of the conversion methods from the GeometryFactory class. Finally, the index package deals with the indexation of spatial data for its faster processing. Details on the way these different capabilities are implemented are given in the next sections.

3.1.1 2D Geometry Types Management

The geometry package of Elcano contains a concrete class for each geometry type described in the ISO 19125 standard. These classes use the JTS spatial library, which is specifically conceived to comply with many ISO standard (including ISO 19125) and the OGC recommendations (Davis and Aquino, 2003). This choice avoids the problems faced by Magellan, which are due to the integration of an inadequate spatial library like stated above. The system could have used JTS classes directly, as Spatial Spark and GeoSpark do, but for optimization purposes, it seemed interesting not to be constrained by the implementation of a chosen spatial library. To this end, the Elcano geometry package uses a JTS-independent class hierarchy by applying the "proxy" design pattern (Gamma et al, 1994). This choice of conception allows also to accelerate, whenever possible, the JTS methods by overwriting them.

3.1.2 Spatial SQL Functions Management

In order to make the spatial functions and operators defined in ISO 19125 available as SQL functions in Elcano, different User Defined Functions (UDF) has been defined. All these functions are in fact shortcuts to the different methods supported by the different geometry types (i.e. classes included in the geometry package) and specified in the ISO 191125 standard. The build() method of the SqlLoader class in the loader package is in charge of declaring all these functions at the initialization stage of the application.

3.1.3 Spatial Data Persistence

Elcano provides a unified procedure for the loading of all 2D geometry types and their persistence. The Table class of Elcano enables the definition of geometric features in WKT. WKT is a concise textual format defined in the ISO 19125 standard. Elcano thus allows to load tabular data (for example from a CSV file where the geometry component of each row is defined in WKT) in the form of an SQL temporary table. The management of more specific formats like JSON (Bray, 2014), GeoJSON or

possible spatial extensions to Big data specific file formats like Parquet (Vorha, 2016) could also be easily added to the system by simply inheriting the Table class.

3.1.4 Data Types Extensibility

The GeometryFactory class implements the "abstract factory" design pattern (Vlissides et al, 1995) and allows the extensibility of Elcano. Other geometry types than those defined in the ISO-19215 standard could thus be added in the future, such as Triangles and TINs in order to manage DTMs (Digital Terrain Models).

3.1.5 Spatial Indexation

The index package of Elcano contains all classes in charge of the spatial indexation of data stored in Elcano. It drastically speeds up all spatial processes. This component is inspired from the one implemented in Spatial Spark but with some hooks for better performance. Its use is illustrated in the benchmark section. Its detailed description is though out of the scope of the present paper. It will be described and detailed later in another publication.

4 BENCHMARK

The present section compares the performances of Elcano with another spatial data management and processing systems using Spark, aka. Spatial Spark. They are also compared with a well-known and widely used classical spatial database management system (DBMS): PostGIS (Obe and Hsu, 2015). Spatial Spark has been chosen among the studied systems that manage spatial indexation because it is the only one that could be extended to support SQL queries (by performing an important reimplementation though). So, it is the only one of the tested prototype that really compares to Elcano. As for PostGIS, it is to our point of view, a reference implementation of the ISO 19125 standard with which we can compare. In addition, it proposes efficient and reliable spatial indexation methods.

For the needs of this benchmark, Elcano and Spatial Spark have been installed on a cluster of servers using a master server with 8 Go of RAM and nine slave servers with 4 Go of RAM. Each of these computers uses the CentOS 6.5 operating systems and height Intel Xeon 2.33 GHz processors. PostGIS has been optimized with the pgTune library (https://github.com/le0pard/pgtune) and tested in comparable conditions.

In each of the 3 tests performed, we count the number of resulting elements from a spatial join between two tables. We group the elements of these tables by pair, according to a given spatial relation, namely the intersection. This spatial relation has been chosen because it implies complex and sometimes time consuming processing. The use of a fast and reliable spatial indexation system is also of importance in such a process. The contents of the tables used in the test is fixed. The management of changing data is out of the scope of the tests.

Test 1 compares the execution time of the three systems with a raise in data volume. It consists in counting the intersections between an envelope around Quebec province and seven sets of points randomly dispatched in an envelope around Canada. These seven sets contain respectively 1000, 10 000, 100 000, 1 million, 100 million and one billion points.

Table 3 presents a synthesis of the first test results for the three studied systems. In order to facilitate their comparison, the duration cumulates the indexation time and the first query time. It appears that performances of Elcano performances are better than those of PostGIS and Spatial Spark beyond one million points. PostGIS is the best choice for lower volumes but encounter a significant slowdown after a certain threshold: it requires many hours to process 100 million points against five minutes for Elcano. The difference between Spatial Spark and Elcano is more tenuous but increases in favor of Elcano as data volume increases.

The drop in PostGIS performances when data volume increases is probably explained by its weak horizontal scalability: this system is not designed for Big data management. In return, performances of Elcano when compared to Spatial Spark can be explained by its usage of Spark SQL. Indeed, the latter uses specific query optimizations and Spark's caching system (Armbrust et al, 2015). But for low data volumes (under one million points), the classical PostGIS solution is better, probably because of its simpler distributed treatments architecture. In a similar way, the best performances of Spatial Spark between one and ten million points can probably be explained by the additional treatments imposed by the use of Spark SQL by Elcano.

Table 3: Test 1 – Processing time with a raise of the data volume.

Volume (points)	PostGIS (ms)	Spatial Spark (ms)	Elcano (ms)
1000	234	6 543	9 516
10 000	326	6 622	9 714
100 000	3 783	8 301	9 030
1 000 000	29 898	8 301	10 747
10 000 000	269 257	20 487	17 099
100 000 000	5 752 821	55 017	37 378
1 000 000 000	More than 10 hours	399 100	273 074

Test 2 compares the horizontal scalability of Elcano and Spatial Spark for a number of servers from one to nine. It compares the count of the intersections between the envelope of the Quebec province and one billion points randomly dispatched in a bounding box of Canada. PostGIS performance is not measured for this test because the previous test clearly underlies its poor performances for large data volumes and there is no way to distribute the processing between many servers as PostgreSQL has not been designed for horizontal scalability.

The table 4 presents a synthesis of the results of this second test. Spatial Spark and Elcano both appear to have a good horizontal scalability. Furthermore, the execution time of the two systems presents a similar drop from one to nine servers: 87,4% for Spatial Spark and 87,2% for Elcano. But Elcano remains approximately 1.5 faster than Spatial Spark regardless of the number of servers.

Table 4: Test 2 – Horizontal scalability when the number of server increases.

Servers	Spatial Spark (ms)	Elcano (ms)
1	3 349 414	2 196 344
2	1 718 672	1 123 153
3	1 143 790	762 536
4	875 284	588 401
5	696 195	473 635
6	586 211	391 297
7	511 111	340 784
8	456 446	314 796
9	423 647	280 761

Elcano's superior brute speed in this second test can probably be explained by its using of Spark SQL. Otherwise the rates of scalability of the two systems are very close, maybe because both rely on the JTS spatial library for the implementation of the spatial analysis algorithms.

Test 3 compares more finely the performances of PostGIS, Spatial Spark and Elcano. It counts the intersections between one million points in an envelope of Canada and the points in a copy of this set. Therefore, a total of 100 billion intersection tests (spatial join) are processed. The execution time is spread between indexation time, first query time (cold start) and second query time (hot start). Hot start queries are more representative of the response times in a running environment in production. Indeed, while indexing is only necessary once for the two given tables, an SQL query must be started for each spatial join operation applied to them.

Table 5 offers a summary of the results for this third test. PostGIS presents a spatial indexation time a bit shorter than Spatial Spark, but the execution time of its first SQL query is then much longer. Elcano presents the best performances in all cases: its indexation time is five time lower than with PostGIS and the execution of its first query is two times faster than with Spatial Spark. Elcano is also the only solution to execute a second SQL query on the same data significantly faster than the first: the second execution is 26 times faster.

The last point can probably be explained by Spark SQL's caching system.

Table 5: Test 3 – Execution time is spread between indexation time, first query time and second query time.

Solution	Indexation time (ms)	First query (ms)	Second query (ms)
PostGIS	29 756	100 742	100 742
Spatial Spark	36 824	36 824	36 824
Elcano	13 578	15 754	1 393

So, to sum up, above a given data volume, Elcano surpasses PostGIS and Spatial Spark in terms of execution speed. It presents a scalability similar to the one of Spatial Spark, but a better execution time when the number of servers increases.

5 CONCLUSION AND PERSPECTIVES

In conclusion, while Big data with a spatial component are in the midst of many scientific, economical and societal issues and while Hadoop has become a mature *de facto* standard for Big data processing, the number of processing and management systems for this type of data using the Hadoop environment and available in the market is limited. All available solutions are only prototypes with limited capabilities. Moreover, only five solutions are managing spatial data from Spark, which is perhaps the most promising Hadoop module for this type of processing, and none of these systems can entirely handle the geometry types and SQL spatial functions specified in the ISO 19125 standard.

To tackle this issue, the present paper proposes a new spatial Big data processing and management system relying on Spark: Elcano. It is based on the SQL library of Spark and uses the JTS spatial library for its compliance with the ISO's standards. Thanks to this approach, all SQL functions and operators defined by the ISO 19125 standard are fully supported.

The proposed model on which Elcano relies is not a simple implementation of JTS. It comes with the possibility to use SQL spatial queries with a data model that can evolve. Furthermore, it integrates the geometric types on a context of Big data and comes with a scalable spatial indexation system which will be detailed in an upcoming article.

In addition, Elcano offers better performances than Spatial Spark and a similar scalability. The detailed study of all the possibilities in term of spatial indexation management remains however to be done. A way to address it could be to adapt the no Hadoop solution defined by (Cortés et al, 2015) to the Spark environment, but there is also many classical spatial data indexation modes that could be explored and adapted in order to fulfill the big data processing requirements.

In a larger perspective, it could be interesting in a near future to enable the management of the elevation together with dedicated data types such as Triangles and TINs in the current model. Raster data types, maybe via the use of RasterWKT, are also considered for inclusion. That would allow to apply the model to many new challenging situations such as the processing of large collection of images coupled with vector data analytics capabilities or the building and analysis of high resolution digital elevation models (DEM) or DTM without being compelled to split them into tiles in order to be able to process them at a whole.

The current version of Elcano manages only batches of data, but adding the possibility of processing and displaying continuously received data (in streaming) could be very interesting (Engélinus and Badard, 2016). Such an extension could indeed enable the design of real time geospatial analytical tools that will help in users (analysts, decision makers, …) in making more informed decisions on more up-to-date data and in a shorter period of time. Furthermore, it could provide some advanced features that deals with the temporal dimension of the data, as for example by excluding all data outside of a defined temporal window (Golab, 2006). Such extensions could allow the modelling of such data as a spatiotemporal event or flow and maybe to dynamically detect "hot spots" (Maciejewsky et al, 2010) in the stream.

But, if Spark can technically handle streaming, taking it into account would induce several conceptual and technical problems. It would be necessary to define a mode of spatial indexation able to manage fluctuating data. Furthermore, what would be the visual variables to use for this type of data in order to represent their dynamic structure? Those defined by Bertin in 1967 (Bertin, 1967) and widely used since are inappropriate because of their strict limitation to a static spatiotemporal context. More recent works have tried to add visual variables to Bertin's models in order to represent motion (MacEachren, 2001; Fabrikant and Goldsberry, 2005), but their application in a context of Big data remains unaddressed. Furthermore, once these conceptual issues are solved, the definition of a system that is effectively able to represent and manage streamed data remains to be done. This could not be a simple add-on to the classic geographic information systems (GIS): they are designed to be efficient for classical data only and are not able to deal with the huge amount of data and velocity that Big data implies. How then is it possible to manage and to represent fluctuating Big data in an efficient way, without losing the horizontal scalability offered by Hadoop? This rich problematic seems to require the definition of a new type of GIS. This will be the bottom line of our future research works.

ACKNOWLEDGEMENTS

We acknowledge the support of the Natural Sciences and Engineering Council of Canada (NSERC),

funding reference number 327533. We also thank Université Laval and especially the Center for Research in Geomatics (CRG) and the Faculty of Forestry, Geography and Geomatics for their support and their funding. Thanks to Cecilia Inverardi and Pierrot Seban for their thorough proof reading and to Judith Jung for her advices in the writing of this paper.

REFERENCES

A. Aji et al, 2013. “Hadoop GIS: a high performance spatial data warehousing system over mapre- duce”. In: *Proceedings of the VLDB Endowment 6.11, p. 1009–1020*.

M. Armbrust et al, 2015. “Spark SQL: Relational data processing in spark”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, p. 1383–1394.*

T. Badard, 2014. “Mettre le Big Data sur la carte : défis et avenues relatifs à l’exploitation de la localisation”. In: Colloque ITIS - Big Data et Open Data au coeur de la ville intelligente. *Québec : CRG.*

A. Eldawy, M. F. Mokbel, 2013. "A Demonstration of SpatialHadoop: An Efficient MapReduce Framework for Spatial Data.″ *Proceedings of the VLDB Endowment.*

J. Bertin, 1967. “Semiologie Graphique: Les Diagrammes, Les Reseaux, Les Cartes”.

T. Bray, 2014. The javascript object notation (json) data interchange format, RFC 7158.

R. Cortés et al, 2015. “A Scalable Architecture for Spatio-Temporal Range Queries over Big Location Data”. In: Network Computing and Applications, *IEEE 14th International Symposium, p. 159–166.*

M. Davis, J. Aquino, 2003. Jts topology suite technical specifications.

J. Dean, S. Ghemawat, 2008. “MapReduce: simplified data processing on large clusters”. In: *Communications of the ACM 51.1, p. 107–113.*

J. Engélinus, T. Badard, 2016. “Towards a Real-Time Thematic Mapping System for Strea-ming Big Data”. In: *GIScience, Montreal.*

E. Gamma et al, 1994. Design Patterns: Elements of Reusable Object-Oriented Software.

A. Eldawy, M. F. Mokbel, 2014. “Pigeon: A spatial mapreduce language”. In: Data Engineering, *2014 30th International Conference on IEEE, p. 1242–1245*.

A. Eldawy et M. F. Mokbel, 2015. “The Era of Big Spatial Data: A Survey”. In: *Information and Media Technologies 10.2, p. 305–316*.

M. R. Evans et al, 2014. “Spatial big data”. In: Big Data: Techniques and Technologies in Geoinformatics, p. 149.

S. I. Fabrikant, K. Goldsberry, 2005. “Thematic relevance and perceptual salience of dynamic geovisualization displays”. In: *Proceedings, 22th ICA/ACI International Cartographic Conference, Coruna*.

S. Fermigier. 2011. Big data et open source: une convergence inevitable? URL: http: //projet-plume.org.

C. Franklin, P. Hane, 1992. “An Introduction to Geographic Information Systems: Linking Maps to Databases [and] Maps for the Rest of Us: Affordable and Fun.” In: *Database 15.2, p. 12–15.*

L. Golab, 2006. “Sliding window query processing over data streams”. Doctorate thesis. *University of Waterloo.*

L. Gu, H. Li, 2013. “Memory or time: Performance evaluation for iterative operation on hadoop and spark”. In: *High Performance Computing and Communications & IEEE 10th International Conference, Embedded and Ubiquitous Computing. 2013, p. 721–727.*

J. N. Hugues et al, 2015. “GeoMesa: a distributed architecture for spatio-temporal fusion”. In: *SPIE Defense + Security. International Society for Optics et Photonics. 94730F.*

ISO 19125-1, 2004. Geographic information -- Simple feature access -- Part 1: Common architecture. ISO/TC 211, 42 pages. URL: https://www.iso.org/standard/40114.html.

ISO 19125-2, 2004. Geographic information -- Simple feature access -- Part 2: SQL option. ISO/TC 211, 61 pages. URL: https://www.iso.org/standard/40115.html.

A. M. MacEachren, 2001. “An evolving cognitive-semiotic approach to geographic visualization and knowledge construction”. In: *Information Design Journal 10.1, p. 26–36.*

R. Maciejewsky et al, 2010. “A visual analytics approach to understanding spatiotemporal hots- pots”. In: *IEEE Transactions on Visualization and Computer Graphics 16.2 p. 205– 220.*

J. Manyika et al, 2011. “Big data: The next frontier for innovation, competition, and productivity”. In: *The McKinsey Global Institute.*

B. Mericksay, S. Roche, 2010. “Cartographie numérique en ligne nouvelle génération: impacts de la néogéographie et de l’information géographique volontaire sur la gestion urbaine participative”. In: *Nouvelles cartographie, nouvelles villes, HyperUrbain.*

R. O. Obe et L. S. Hsu, 2015. PostGIS in action. *Manning Publications Co.*.

Commonwealth Computer Research, 2017. Apache Spark Analysis. URL: http://www.geomesa.org/documentation/tutorials/spark.html.

S. Ram, 2015. Magellan: Geospatial Analytics on Spark. URL: http://hortonworks.com/blog/magellan-geospatial-analytics-in-spark/.

R. Sriharasha, 2016. Magellan’s Github - issue 30. URL: https://github.com/harsha2010/magellan/issues.

J. Vlissides et al, 1995. “Design patterns: Elements of reusable object-oriented software”. In: *Reading: Addison-Wesley 49.120, p. 11.*

D. Vorha, 2016. “Apache Parquet”. In: *Practical Hadoop Ecosystem. Springer, p. 325–335.*

T. White, 2012. Hadoop: The definitive guide. *O'Reilly Media, Inc.*

D. Xie et al, 2016. Simba: Efficient In-Memory Spatial Analytics. URL: https://www.cs.utah.edu/~lifeifei/papers/simba.pdf.

S. You, et al, 2015. "Large-scale spatial join query processing in cloud". In: *Data Engineering Workshops (ICDEW), 31st IEEE International Conference, p. 34–41*.

J. Yu, 2017. GeoSpark's Github- issue 33. URL: https://github.com/DataSystemsLab/GeoSpark/ issues.

J. Yu et al, 2015. "Geospark: A cluster computing framework for processing large-scale spatial data". In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, p. 70.*

M. Zaharia et al, 2010. "Spark: cluster computing with working sets". In: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. T. 10, p. 10.*