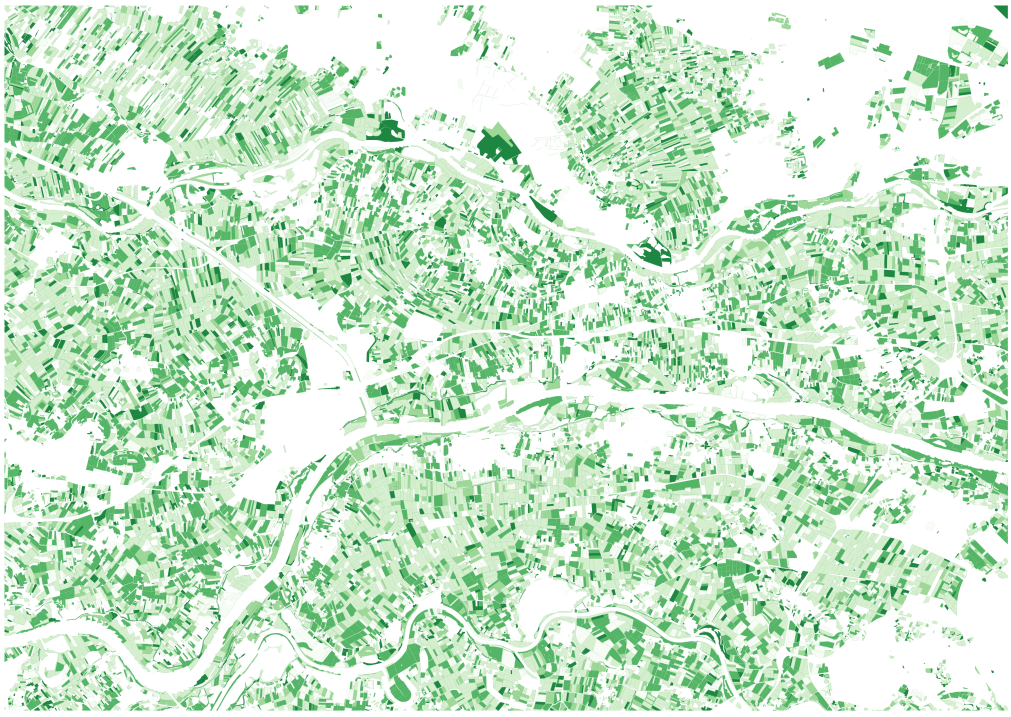
Centre for Geo-Information

Thesis Report GIRS-2018-06

Managing Big Geospatial Data with Apache Spark
MSc Thesis

Hector Muro Mauri



March 2018

WAGENINGEN
UNIVERSITY & RESEARCH

Managing Big Geospatial Data with Apache Spark
MSc Thesis

Hector Muro Mauri

Registration number

920908592110

Supervisors:

Dr Arend Ligtenberg

Dr Ioannis Athanasiadis

A thesis submitted in partial fulfillment of the degree of Master of Science at Wageningen University and Research Centre, The Netherlands.

March 2018

Wageningen, The Netherlands

Thesis code number: GRS80436

Thesis Report: GIRS 2018 06

Wageningen University and Research Centre

Laboratory of Geo-Information Science and Remote Sensing

Contents

1 Introduction

1.1 Introduction & Context

Over the last decade Big Data has become a buzzword used to define probably too many things. What is undeniable is that the amount of data generated every day has not stopped increasing as well as the source types where this data comes from. Estimations report that 2.5 quintillion bytes of data are generated every day and a large portion of it is geographical location related [Lee and Kang, 2015]. NASA's Earth Observing System only, generates 1 terabyte of data every day [Leptoukh, 2005].

The large portion of data that is location aware is commonly named Geospatial Big Data [Lee and Kang, 2015, Li et al., 2016]. Geospatial Big Data can be defined as the data whose "size, variety and update rate exceed the capacity of commonly used spatial computing and database technologies to learn, manage and process data with reasonable effort" [Shekhar et al., 2012]. This is only possible due to a shift in the way Geospatial data is collected. Whereas it used to be based on technically demanding, accurate, expensive techniques, where the measuring process was itself sometimes an art [Lee and Kang, 2015], nowadays there is less accuracy in certain aspects and the problem is no longer how or where to get the data from, but what to do with it.

Much research has been done regarding Big Data Analytics, but less in terms of Geospatial Big Data Analytics. The most common is related to location-based business or social media analysis, [Zeng et al., 2010, Phillips et al., 2010] are examples of it, due to the economical potential and its huge expansion in the past 10 years. This does not mean though that Geospatial Big Data Analytics are constrained or only meant for location-based analysis at a user level.

The exponential growth of data globally is undeniable, either Geospatial or not, and the fact that it will not stop, but increase more and more. Thus, the problem that arises is: how to deal with it?

Taking into account that a single processor is not enough to cope with Big Data, it has been necessary to shift to other paradigms, such as cloud computing. Big Data is a new hype concept to many of us is probably because before 2005 the general idea was that with every new generation of processors the processing speed incremented as well, but later in the same decade it was seen that the Moore's law [Schaller, 1997] is only achieved by increasing the number of cores / nodes, since the clock cycles in the processors stopped increasing (Figure 1).

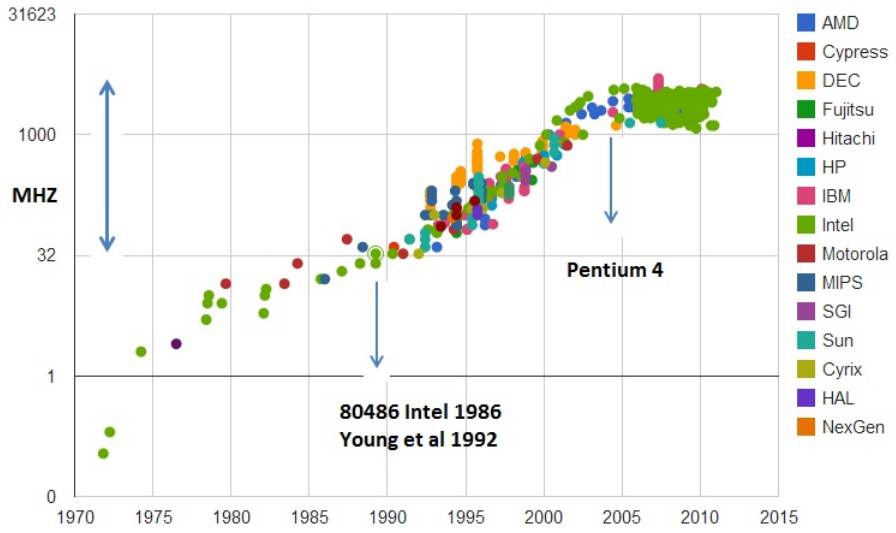


Figure 1: Clock CPU Scaling [Cook, 2013]

When this situation was encountered, research head for new methods, such as parallel distributed paradigms

[Lee et al., 2014, Shekhar et al., 2012] as the best solution to deal with big data sets so that the data can be split, sent to multiple processors and retrieved in such a way that it can be managed and processed.

The concept of parallel programming is not a new, but parallelizing implies a complete different mindset in terms of software development. The biggest obstacle encountered is because of the non-determinism[1] caused by concurrent actions accessing a shared mutable state.

A solution for parallelizing has traditionally been functional programming. Pure[2] functional programming means coding without mutable variables, assignments and other imperative control structures. Functional programming solves the non-determinism problem, also known as data races[3], by avoiding mutable states or, in other words, strictly controlling the simultaneous access to mutable data.

However, dealing with Big Data sets does not only reduce to a programming language scope, but an entire system capable to deal with parallel distributed problems and algorithms is needed. Hadoop MapReduce was one of the first platforms [Padhy, 2013, Hadoop, 2017] to appear. However, the best, simpler and more complete framework at the moment is Apache Spark [Shanahan and Dai, 2015, Spark, 2017a].

Apache Spark is a cluster-computing platform that provides an API for distributed programming. It is by far the most used platform thanks to its ease of use. It can run on different cluster types (e.g. Hadoop), clusters managers (e.g. YARN) and it has different programming API's in Scala, Java, R and Python [Spark, 2017a]. While performance is key when dealing with Big Data, the fact that there are so many API's allows many scientists and researchers to use it. The Apache Spark Stack (Figure 2) offers such interoperability.

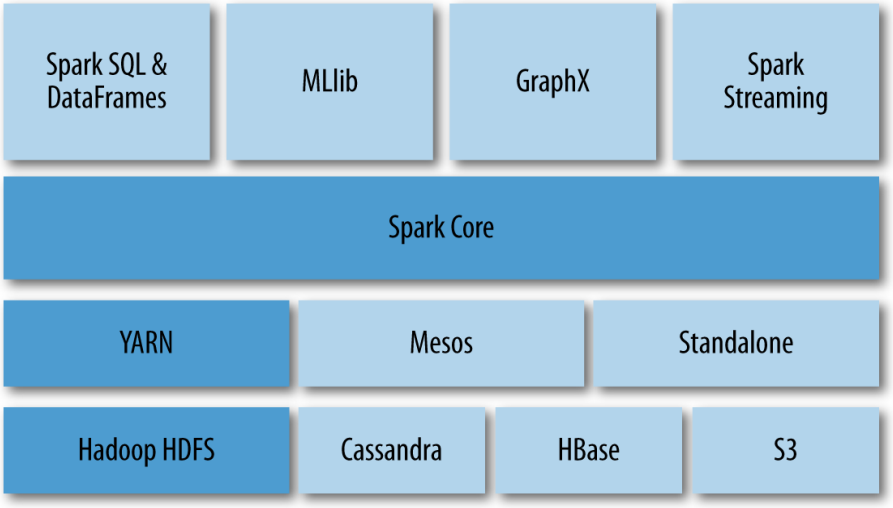


Figure 2: Apache Spark Stack [Bengfort and Kim, 2016]

From a more technical perspective, Apache Spark ensures the *scalability* and *fault tolerance*, which are key aspects in Big Data. On the one hand, scalability stands for the ability to adapt, so that any algorithms can still run no matter how big the data set is (vertical scalability) or how many processors are used (horizontal scalability). On the other hand, fault tolerance means that, in case a node in the system would fall, no data would be lost.

Both key aspects are achieved on the base of Apache Spark's distributed memory abstraction, Resilient Distributed Data sets (RDD) [Zaharia et al., 2012]. RDD's are what is known as data types in traditional programming, but for parallel distributed programming. Fault tolerance is achieved thanks to a *lineage* and *immutability*. Immutability defines that any *transformation* applied to a file (RDD) will create another file. Thus, in case of

[1]Non-determinism: A mode of computation in which, at certain points, there is a choice of ways to proceed which cannot be predicted in advance [Oxford English Dictionary, 2007].

[2]Many programming languages are non-pure functional or flexible.

[3]A data race occurs when two or more threads in a single process access the same memory location concurrently [Oracle, 2017].

failure, all or a part of transformations can be reapplied by simply looking back at the *lineage* or the "history" of transformations [Bengfort and Kim, 2016].

One last key aspect that has put out many features and libraries is Apache Spark's Catalyst optimizer [Armbrust et al., 2015]. Thanks to this extensible optimizer in the core of Apache Spark, applications and libraries can be written on top of Apache Spark and still take advantage of all the benefits that the core engine provides.

Finally, parallel distributed programming ensures a faster time of calculation and a bigger capacity in terms of data volume, but it comes at a cost. Since the data is organized in a completely different way than traditional programming, the algorithms must adapt as well, so that the same process can be done. Therefor, how can geospatial algorithms be parallelized.

Research suggest that it depends on the nature of the problem itself [Griffith et al., 2017], since many of the geographical problems are easily parallelized or have the "parallel property", where a large problem involving n locations can be subdivided in n smaller problems [Guan et al., 2011]. It becomes more difficult when partitioning the space is not as easy as a regular grid [Guan and Clarke, 2010], but it is definitely possible and both research and industry are more often looking towards "thinking in parallel" [Turton and Openshaw, 1998].

1.2 Problem definition & Research questions

In the last couple of years, a few software packages that run on top of Apache Spark have come out into the open as GeoSpark [Lenka et al., 2016] or Magellan [Vasavi et al., 2018]. There already exist a few promising use cases where Geospatial Analysis has been conducted using Apache Spark packages, such as finding out spatial relations in Taxi and Limousines trips in NYC [Sriharsha, 2017] or how Hurricane Harvey affected the supply chain of vessels visiting ports in the Gulf of Mexico in late summer 2017 [Hezbor and Hughes, 2017].

While specific examples have been reported in recent literature, where parallel distributed Geospatial software packages were used, it is unclear whether these libraries are mature enough to be used in a broader scope. It is obvious that the level of development can not be compared to traditional GIS tools, because the latter exist for decades already. This does not mean that Apache Spark packages should not be taken into consideration either.

In order to find an objective way to qualify the maturity of these software packages, a benchmark test is the most suitable idea. Thus, the aim of this research is to answer the following question:

- What is the maturity of Geospatial Big Data software packages in Apache Spark? This creates the following subquestions:
  - How can the maturity of Geospatial Big Data software packages in Apache Spark be assessed?
  - What is the performance of Geospatial Big Data software packages in Apache Spark compared to traditional Geospatial software packages?
  - Can Geospatial Big Data software packages in Apache Spark be used in a production level?

1.3 Structure of the document

First, the methodology followed to answer the research question is described as well as the tools used to develop such methodology. Then the design of the different tests used in this research are presented. Later the results of applying such tests are shown followed by the discussion of them. This reports ends with the conclusions that have been extracted from conducting this research together with some recommendations for the future.

2 Methods & Tools

2.1 Methods

The maturity of the software packages will be assessed in four steps as can be seen in Figure 3. The first thing that will be described is the functionality of the software packages, which operations do they support. Then, with a clear picture of the functions the software packages can perform, they will be put into practice and compared against traditional software by means of a performance test. Later on, with the functionality and performance results, the software packages will be used in a use case. Finally, the self experience of this research will be also added to the final maturity description.

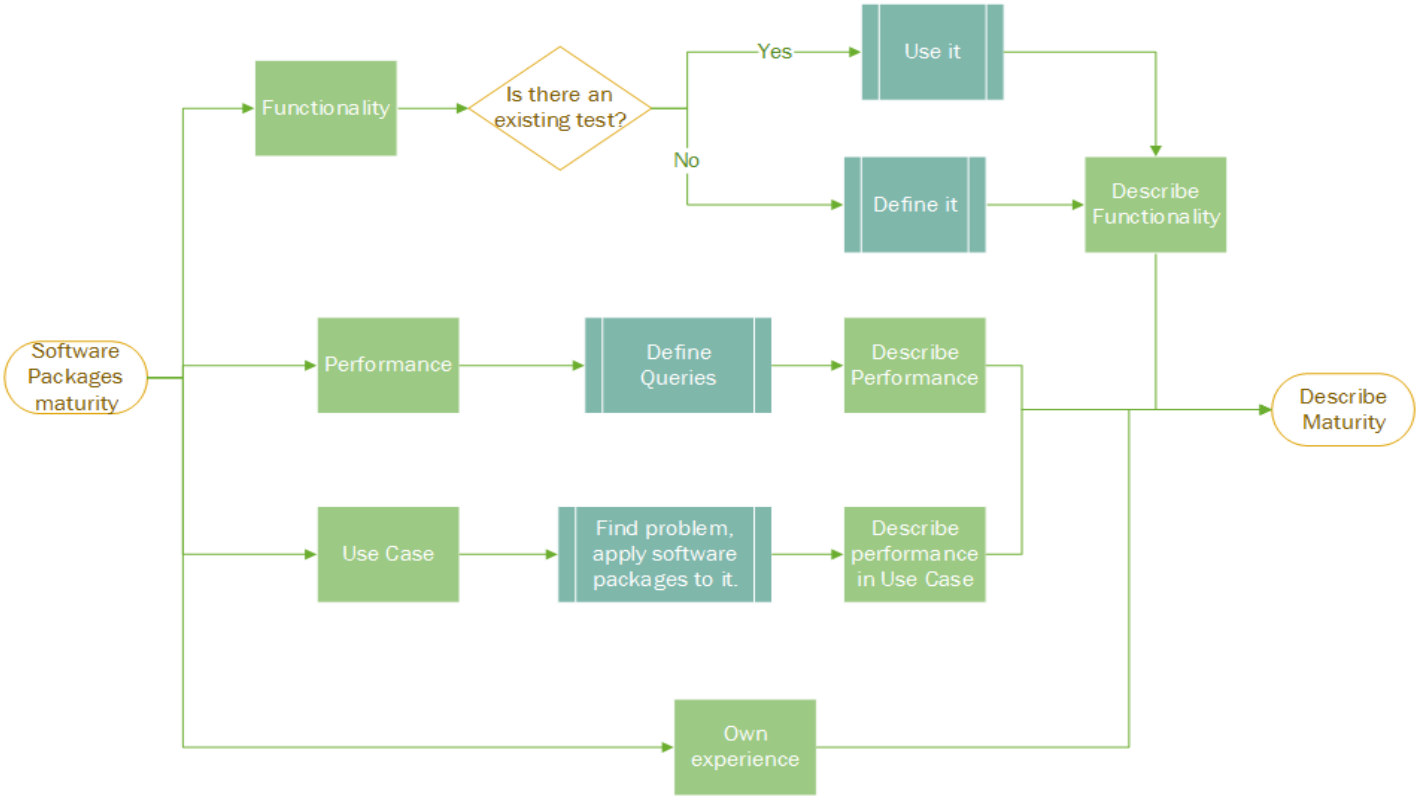


Figure 3: Methodology Flowchart

2.1.1 Functionality Test

The first step is to describe the software packages' functionality. In order to do that in a quantifiable and objective way, a benchmark test will be defined. This will be done by means of a literature review on both the basic operations within the GIS scope and the already existing Spatial Benchmark tests. Based upon that it will be decided on whether make use of an existing benchmark test or define one for this research, based on already existing tests.

2.1.2 Performance Test

After describing every library's utility, they will be put into practice. This will be done by means of defining a set of queries that will test the vast majority of the functionality. The execution time will be taken as the performance measurement. These measurements will be compared against the ones of performing the same set of queries in a spatial RDBMS.

2.1.3 Use case

The next step will consist on making use of the software packages in an environment where there is a spatial data management system in production. This part will consist in replicating one or more situations that often occur in the environment chosen in order to compare a traditional system with an Apache Spark one.

2.1.4 Own Experience

Finally, the own experience of the conductor of the research will be taken into account. The potential users of these types of tools are GIS analysts that do not have a deep knowledge in Software Engineering. Then, further aspects beyond technical issues will be taken into account, such as the documentation quality, if existing, installation process or its speed of development.

2.2 Tools

The software packages to be tested in this research have been chosen based on some conditions. Due to time constraints the number of software packages to be tested has been limited to four. At the same time, the chosen software packages must be developed on top of Apache Spark. This decision is based on the scientific community shift towards this specific tool regarding parallel distributed management tools. Finally, the existence or not of publications where the software packages are used to solve a use case, has also been taken into account in order to have an indication of its potential use.

2.2.1 Magellan

Magellan [Magellan, 2017] is a distributed execution engine for geospatial analytics on big data and it is implemented on top of Apache Spark. The developers are active parts in Databricks[4]. It is the first library to extend Spark-SQL to provide a relational abstraction for Big Geospatial analytics. In such a way, SQL queries, or SQL alike queries, can be performed in Data Frames to evaluate geometric expressions, while the engine (Spark-SQL) takes care of efficiently laying data out in memory during query processing, picking the right query plan, optimizing the query execution with cheap and efficient spatial indexes. [Sriharsha, 2017, Spark, 2017a]

Magellan is part of the Apache Spark packages [Spark, 2017b] and can also be found in different Maven repositories [Maven, 2017]. Magellan's only focus is vector data so far and defines three predicates: Within, Contains and Intersects. Finally, documentation does not yet exist, besides a few incomplete explanations that can be found in its Github repository and a couple of entries in a blog [Sriharsha, 2017] there is no proper documentation. There is an example on a big data set of taxi trips in NYC.

2.2.2 GeoPySpark

GeoPySpark is being developed at Azavea, which is part of Location Tech [Azavea, 2017, Loation Tech, 2017]. GeoPySpark is actually a Python binding library for GeoTrellis, which is developed in Scala. By using PySpark, it is able to provide an interface into the GeoTrellis framework using Python. The main reason why a Python binding is being developed is because "GeoTrellis has a limited user base due to the geospatial community's preference for other languages such as Python and R" [GeoPySpark, 2017].

GeoPySpark is mostly dedicated to Raster Data only. This includes basic operations such as Map Algebra, but also some distance operations, such as cost analysis, and transformations. As of today, there are plenty tutorials

[4]Databrics is a company founded by the creators of Apache Spark, that aims to help clients with cloud-based big data processing using Apache Spark [Databricks, 2017a].

and examples, in the form of python notebooks, that can be found in their Github repository ([GeoTrellis, 2017]). At the same time, documentation is constantly updated and published in their website ([GeoPySpark, 2018]).

2.2.3 GeoSpark

GeoSpark is a Java software project created at Arizona State University's Datasylab, together with Babylon, a GeoSpark visualization extension. Its main goal is to provide a full-fledged cluster computing framework to load, process and analyse large-scale spatial data in Apache Spark [Geospark, 2017].

GeoSpark is solely dedicated to Vector data. It has different methods for partitioning and indexing spatial data, which adds flexibility depending on the data in use [Huang et al., 2017b]. It has a set of Java-like built-in functions to perform spatial queries, but also a set of pre-defined PostGIS-like queries.

GeoSpark extends Apache Spark's RDD's to Spatial RDD's to benefit from all the parallelization power with a spatial property. Recently a connector to Spark SQL has been added. This allows benefiting from Apache Spark's Data Frames abstractions, with a spatial component, which reminds of how Magellan does it. Finally, it is important to mention that "GeoSpark uses the real shape of a geometry to calculate the result which might be slow. When using spatial tree index (Quad-Tree or R-Tree), GeoSpark always follows filter-refine model to ensure the correct results" [Yu et al., 2016]. Most of the existing libraries only use MBR (Minimum Bounding Rectangle), or they do not follow filter-refine model to refine their results after querying the tree-index.

2.2.4 GeoMesa

GeoMesa is an open-source, distributed, spatio-temporal database that allows large-scale geospatial analytics on cloud and distributed computing systems. GeoMesa, as well as GeoPySpark or GeoTrellis, is being contributed by Location Tech [Loation Tech, 2017], as well as CCRi [CCRI].

One of the key aspects of GeoMesa, as depicted in their website, is the interoperability that it has with a great variety of Data Stores and File Systems, such as Apache Accumulo [Apache Accumulo, 2017], Apache HBase [Apache HBase, 2017] or Apache Cassandra [Apache Casandra, 2017]. It also provides support for near real time stream processing by using the Apache Kafka messaging system [Apache Kafka, 2017]. Finally, a whole range of PostGIS-like queries in order to perform complex geospatial analytics tasks in a simpler way [Hezbor and Hughes, 2017].

There already exist a few use cases that benefit of GeoMesa to process a large quantity of spatial data, such as heat maps from maritime traffic [Hezbor and Hughes, 2017] or an animated visualization for the Superbowl 2015 Tweets [GIS Lounge, 2015] that demonstrate the capabilities of GeoMesa regarding GeoSpatial Big Data analytics

3 Tests Design & Definition

3.1 Functionality Test

3.1.1 Literature Review

Evaluating software is something that has been carried on for decades. Most of the research is focused on how to measure the software quality [Jones, 2008, Fenton and Bieman, 2014, Coleman et al., 1994]. NASA provides with an example of it with the 9 *Technology Readiness Levels* [Mankins, 1995]. These levels are used to define the maturity state of a technology, being the first level the most basic principles and the ninth "Actual system “flight proven” through successful mission operations".

Unfortunately, there is no widely used standard spatial benchmark [Ray et al., 2011]. Some research has been narrowed towards the evaluation of spatial indexing. There has not been research focused on which operations or algorithms should a GIS tool conceive [Myllymaki and Kaufman, 2003, Gurret et al., 1999].

In GIS any possible operation could be into Data exploration or exploratory statistics, Vector and Raster. Much research suggests these three blocks as the best solution to subdivide GIS operations. [Chang, 2004], for example, classifies the basic GIS analytic operations in the following blocks:

1. **Data exploration**
2. **Vector Data Analytics**
3. **Raster Data Analytics**

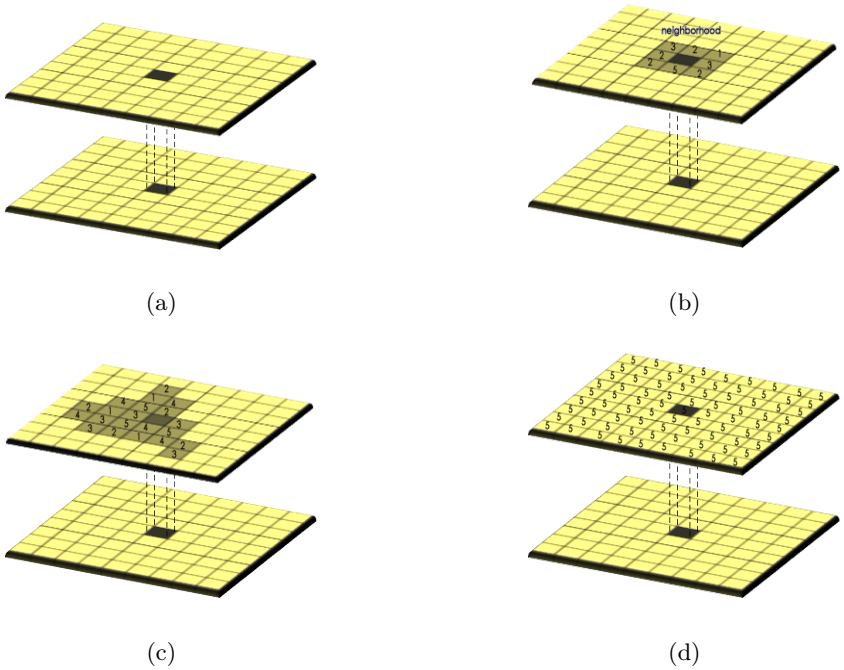
Data exploration is the starting point before any other kind of operation is going to be performed. During explorations, descriptive statistics give an overview of the data, which is the base for further analyses. When performing data exploration to geospatial data sets it only varies from non-geospatial data sets in the obvious fact that the latter does not involve both spatial and attribute data.

Vector data refers to the data that uses coordinates to create spatial features such as points, lines and polygons. One of the key aspects in Vector Data Analytics is the topology [Molenaar, 1998]. Topology in GIS is the spatial relationship between features or objects. This is very important because is what defines the relative location of an object and what ultimately defines any kind of operation on geospatial data. Topology serves as well to create better quality control and greater data integrity.

There is a big range of operations regarding vector data, but a set of basic tools could be the following:

1. **Buffering**: Operation based on the concept of proximity. It creates two surfaces; one within a specified distance from the feature and the other beyond that distance. The buffer area is the one within the two boundaries.
2. **Overlay**: Operation that combines the geometries of two spatial data sets. Output must be the intersection of the inputs with both attributes on it. Example methods are: Union, Intersects, Symmetrical Difference or Identity [Chang, 2004].
3. **Distance Measurement**: Measuring straight (Euclidean) lines between features, geodetic distances or spheric distances.

Finally, Raster and its basic operations are illustrated in [Chang, 2004]. Rasters refer to those datasets in form of a regular grid or pattern. In a raster data set, each cell represents a value at a given location. The value can represent different things; Digital Numbers (DN), height, precipitation volume, temperature, etc. One of the reasons for a very extended raster usage is the benefits that come with regular patterns when storing data and



(a) (b)

(c) (d)

Figure 4: Local (a), Focal (b), Zonal (c) and Global (d) Map Algebra [GIS Dictionary ESRI, 2015]

computing. This is, when data is organized in regular grids, it can be accessed easily since the data location can be known beforehand and does not change.

Raster Data Analytics can be performed in four different levels, also known as Map Algebra[5]:

1. **Local** (Figure: 4a): This is the core of the raster analysis [Molenaar, 1998]. These operations happen at a cell level, the value in the output raster is at the same location on the input raster. Examples of cell operations are [GIS Geography, 2017]:
   - Arithmetic operations: sum, subtraction, multiplication, division.
   - Statistical operations: minimum, maximum, mean, median, etc.
   - Relational operations: greater than, equal to, etc.
2. **Focal** (Figure: 4b): It involves a central cell and a set of surrounding ones. Used mostly for filtering and convolution. Moving windows are also examples of zonal operations.
3. **Zonal** (Figure: 4c): In this type, the new cell value is based on a function of different cell values from different rasters (a minimum of two) based on clustering. The input zones can be contiguous (cells spatially connected) or non-contiguous (cells with the same value) [Ligtenberg, 2016]
4. **Global** (Figure: 4d): This type of operations are those that involve the whole raster. Euclidean distances or cost distances are examples of global operations in raster.

One last set of basic operations that apply to both types of spatial data are geostatistics, i.e. operations that relate to spatial correlation, estimation (Kriging) and simulation. Geostatistics are broadly used in many situations that involve sampling and that require analyses and prediction based on their location [GIS Dictionary ESRI, 2015].

[5]A language that defines a syntax for combining map themes by applying mathematical operations and analytical functions to create new map themes. [GIS Dictionary ESRI, 2015]

Another broadly used geostatistics method is what is commonly known as "Hot Spot Analysis". This type of analysis uses vectors to identify the locations that are statistically significant as "hot spots" or "cold spots" [Lu et al., 2012].

Finally, some publications [Chang, 2004] mention Map Manipulation as a basic GIS tool. It refers to the usage of a GUI to perform operations. Even though having a GUI can be very handy to reach a broader audience, it is beyond the scope of this research. In all, this summarizes the most basic operations within a GIS system.

Based on these tools and more complex ones, after years of development in GIS software, different GIS applications appeared that could reach a broader audience. Those applications include desktop [QGIS, 2016, gvSIG, 2015, Marble, 2014, GRASS Development Team, 2015, Harris Geospatial, 2016, ESRI, 2016, Autodesk, 2015, Bentley, 2017], both open source or proprietary software and GIS libraries and Geospatial Data Management tools [Geo Tools, 2016, PostgreSQL, 2015, GDAL, 2016].

The emerging of many platforms dedicated to the same end, raised the need of a reference test which a Geospatial tool could be tested against.

One of the first Geospatial Data Benchmark test defined was SEQUOIA 2000 [Stonebraker et al., 1993] and its main characteristic was the fact that it was only tapered at Earth Sciences and focused on raster data. At the same time, this benchmark only specifies the queries, and rules for reporting price/performance analysis.

After SEQUOIA 2000, more spatial data benchmark tests research came out. One example is Jackpine [Ray et al., 2011], which is a benchmark test defined at the University of Toronto based on SEQUOIA 2000 to evaluate spatial database performance. Its main goals are: to be able to support a great number of different databases and to include a comprehensive set of workloads. The test is divided in two smaller benchmarks:

1. **Micro Benchmark**: This benchmark has the goal to test the basic topological relationships and spatial analysis functions. So if this is passed, the Spatial Data Base would provide minimal coverage.
2. **Macro Benchmark**: The Macro Benchmark scenarios attempt to model some of the new emerging geospatial web services. Each scenario consists of a set of queries that make up each of the scenarios.

Another example in Spatial Data Benchmark tests is VESPA (VEctorial SPAtial) [Paton et al., 2000]. This benchmark test was created due to the necessity of a benchmark for vector spatial data bases. In the test, it is stated that a benchmark test should always satisfy the following requirements:

- **Ease of use**. It should not be time consuming since it is only meant to be a test.
- **Wide ranging functionality**. It should asses a great number of features.
- **Scalability**. The data sets generated should be as small or as big as needed.

In order to achieve a wide ranging functionality, VESPA consists of a set of 26 tasks, aggregated in ten different groups:

- **Updates**: Insertion and deletion of objects.
- **Set operations**: Union polygons.
- **Containment operations.**
- **Overlap operations.**
- **Intersect operations.**
- **Adjacent operations.**

- **Search inside area**: Buffer search and window search.
- **Measurement operations**: Size, lengths, areas and Euclidean distances.
- **Spatial Analysis operations**: Aggregation.
- **Non-spatial operations**: Selection, join, aggregation.

It has been decided to define a Functionality Benchmark test because there is no test that adapts to the particular needs of these software packages. The defined test, is based on the methodology approach described in VESPA, but with the basic operations stated in Chang's and Molenaar's approach [Chang, 2004, Molenaar, 1998] on what the basic operations should be for any GIS tool. This responds to the need of creating a Functionality Benchmark test easily reproducible that takes into account the youth of these software packages.

3.1.2 Functionality Benchmark Test Definition

The operations that are part of this test have been divided in an incremental order, based on their complexity. For every operation, the final score is given based on the actual coverage of that operation. It could be the case that one operation is supported, but not for all the possible geometry pairs. This aims to represent a clearer picture of the actual state of development of those libraries. All operations count the same to the final score of the test. The test can be seen in Table 2.

In the next paragraphs, the different classes of operations contained in the test will be explained:

- **Reading Data**

  Before performing any kind of operation, data needs to be retrieved. In this first section of the test, the different file formats that every software package states to be able to perform with, will be tested and a description of the process will be given.

  In Table 1, five vector and raster file formats have been considered as the basic file formats that should be accepted in any GIS tool. They have been selected either for its extended use (e.g. Shapefile, GeoTIFF), for its readiness (e.g. Geo(JSON)) or for its relation with the big data scope (J2).

Vector	**Raster**
Text Files	Text Files
XML	GeoTIFF
WKT	Jpeg
GeoJSON	J2
Shapefile	JSON

Table 1: Basic File Formats

- **Queries**:

  A query or request is the most basic action that can be performed to a data set, whether is spatial or not. It is a subset from the entire data set, based on conditions defined by the user. Querying does not imply any change or modification in the original data set or its meaning. This type of operations have been inherited from the traditional Relational Database Management Systems (RDBMS).

  Queries can be subdivided into:

  – **Attribute Data queries**: This type of queries can be split, at the same time, into simple and complex queries. Simple queries correspond to those requests that only involve one table, whereas complex refer to those that involve two or more tables and the usage of joins.

- **Spatial Data queries**: Feature selection based on spatial relationships and how spatial joins are applied. In this sort of query, topology plays a crucial role [Molenaar, 1998] and its correct and precise management will be tested as well.
- **Raster Data queries**: Raster selection based on raster attributes.
- **Temporal Data queries**: This type of queries is also known as "versioning". Sometimes entities change through time, but it is necessary to keep all the versions. GIS has not been good at this traditionally, but this topic has become of more relevance.

- **Transformations**:

  Transformations are operations that modify the data without changing its thematic and/or geometric meaning. Transformations can be applied to data sets of the same data type or to change from one to another.

  The first group includes the coordinate reference system transformations. This sort of transformations are crucial in any GIS system. A common reference system must exist among all data sets that take part in any operation so that it can be performed satisfactorily. It is not strange that data sets that come from different sources are referenced in different systems. Consequently, transforming different reference systems is an important tool [Robinson, 1960].

  The second kind of transformation refers to the transformation from vector to raster and vice versa. This type of transformations may occur when particular operations of a data type are needed and data sets are not of that data type.

- **Alterations**:

  Alterations are the most complex spatial operation. Alterations modify the thematic and/or geometric definition of the original data set. They imply combining the original files into another one in order to obtain the information needed. Multiple operations across the data occur and this is why alterations are very time-consuming.

  There are many operations within the block of alterations, but the most basic and common ones that have been considered are the following:

  - **Buffering**: Returns a polygon covering all area within a given distance from the input geometry. The distance can be both constant or variable (cost analysis) [Open Geospatial Consortium, 2011].
  - **Intersects** (Figure 5): The topological integration of two spatial data sets that preserves features that fall within the spatial extent common to both input data set [GIS Dictionary ESRI, 2015, Open Geospatial Consortium, 2011].
  - **Within** (Figure 6): Returns the point, line, or polygon features (or portions of these features) that are within the boundaries of polygons in another layer [GIS Dictionary ESRI, 2015, Open Geospatial Consortium, 2011].
  - **Contains**: A spatial relationship in which a point, line, or polygon feature or set of features is enclosed completely within a polygon [GIS Dictionary ESRI, 2015, Open Geospatial Consortium, 2011]. (Contains and Within are "inverse" from each other).
  - **Union** (Figure 7): A topological overlay of two or more polygon spatial datasets that preserves the features that fall within the spatial extent of either input dataset; that is, all features from both datasets are retained and extracted into a new polygon dataset [GIS Dictionary ESRI, 2015, Open Geospatial Consortium, 2011].
  - **Map Algebra**: Operations that can be applied to one or more raster in different levels:
    - **Local**: Set of operations that occur at a cell level.
    - **Zonal**: Set of operations that occur in the surrounding area of a cell.
    - **Focal**: Set of operations that occur in areas. Those can be contiguous or non-contiguous.
    - **Global**: Set of operations that involve the whole raster.
  - **Distance operations**: This is the last set of alterations. It includes the calculation of Euclidean distances (shortest separation between features) and cost distances.

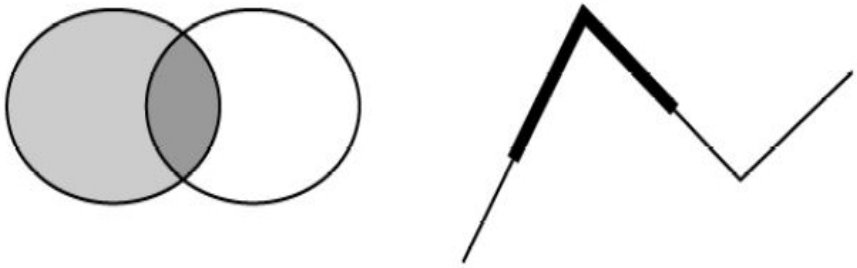


Figure 5: Intersects operation with different vector data types [Open Geospatial Consortium, 2011]

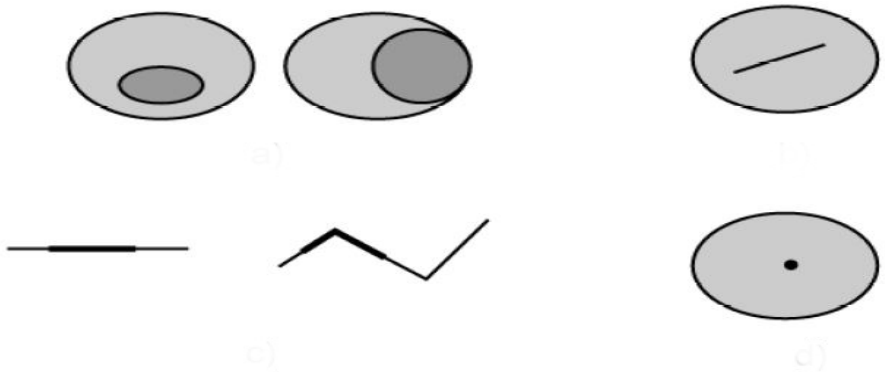


Figure 6: Within/Contains operation with different vector data types [Open Geospatial Consortium, 2011]

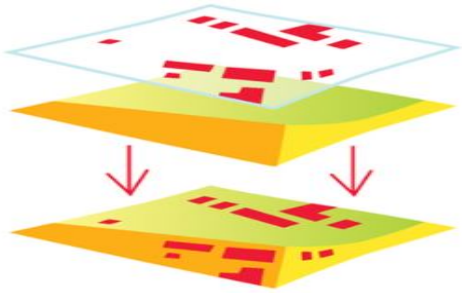


Figure 7: Union of two layers [GIS Dictionary ESRI, 2015]

- **Geostatistics**:

  Geostatistics are a subclass of statistics used to analyze and predict the values associated with spatial or spatiotemporal phenomena [GIS Dictionary ESRI, 2015]. It is applicable to any function distributed in real space. One of its principles, satial correlations, is based on Tobler's first law of geography: "everything is related to everything else, but near things are more related than distant things".

  Geostatistics are widely used in many areas of science and engineering, such as mining, environmental sciences or catastrophe-event management just to mention a few examples.

Operation			Description
Reading Data			Accepted file formats
Queries	Selection	by attributes	Thematic Data Query
			Raster Data Query
	Spatial		Geometric Data Query
	Temporal		Temporal Data Query
Transformations	Coordinate System		Transform from a coordinate system to another
	Vector -> Raster		Convert a vector file into raster
	Raster -> Vector		Convert a raster file into vector
Alterations	Buffering		Create a buffer in a point, line and polygon
	Intersects		Feature intersects feature
	Within		Feature within feature
	Contains		Feature contains Feature
	Union		Union of two features that share spatial extent
	Map Algebra	Local	Operations that occur at a cell level
		Zonal	Operations that occur in a small area around a central cell
		Focal	Operations that occur in areas that share attributes
		Global	Operations that occur in the whole raster
	Distance Operations		Area, length, Euclidean Distance, Cost Distance
Geostatistics			Spatial interpolation (Kriging) Hot Spot Analysis
Total			

Table 2: Reference table for functional test

3.1.3 Functionality Benchmark Test Datasets

The functionality benchmark test will be conducted in all libraries using the same data sets. In order to facilitate validation, a set of simple geometries and grids will be constructed.

In the case of vector data, this means points, lines and polygons manually created using the libraries constructors. With such controlled datasets it can be easily depicted whether the software packages implementations produce the correct results or not. Figure 8 shows an example of the possible topological relationships for polygons. They are defined as squares and contain:

- A polygon completely inside another polygon.
- A polygon intersecting another polygon.
- A polygon in the border of another polygon.

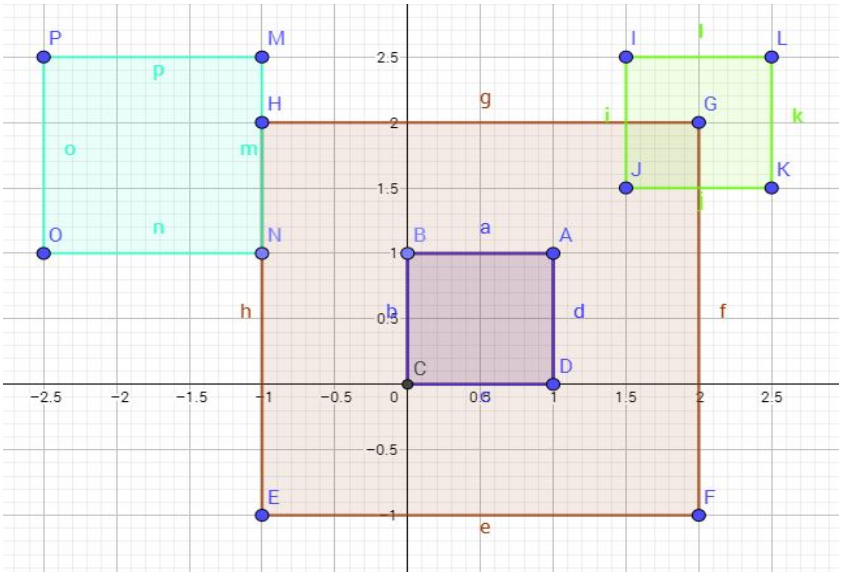


Figure 8: Example of topological relations within polygons.

For the raster data, a squared matrix 4 by 4 filled with 1's will be defined as the dataset.

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

Figure 9: Example matrix for raster data set.

3.2 Performance Benchmark Test

In order to measure the performance, a set of queries has been defined, based on the functionality test and its results, to be tested in the new software packages as well as in traditional software. For the latter, PostgreSQL with its spatial extension PostGIS will be used. At the beginning it was intended to define two different performance tests; one for vector data and the other for raster data. Unfortunately, due to time constraints, only a vector data performance test has been defined.

In the next paragraphs, the different set of queries are explained. For a clearer understatement all queries have been written in SQL or pseudo-SQL, although this does not mean that the real code looks like that. Except for the ordering in returning the results, all queries in all platforms perform the same way. Finally, the number of results has been specified to give an idea of the query size. The operations have been subdivided in the same way as in the Functionality Benchmark Test.

The datasets used for this test are explained later in Section 3.2.4

3.2.1 Queries

Attribute Query: Although this type of query does not involve spatial operators, it is broadly used, necessary and will serve the purpose of seeing Apache Spark's performance with different types of datasets. The following queries have been subdivided into two groups.

The first one contains 4 queries to a text file (csv), where the first two queries are performed to a 65000 records text file and the other two queries are performed to a 12000000 records text file.

No.	Query	No. of Results
1	SELECT * FROM trips_split WHERE trip_distance > 4.5	10141
2	SELECT * FROM trips_split WHERE passenger_count > 2	9670
3	SELECT * FROM trips WHERE trip_distance > 4.5	1910474
4	SELECT * FROM trips WHERE passenger_count > 2	1823993

Table 3: Text File Attribute Queries

In the second group, there are two queries to two different Shapefiles [ESRI, 2016]. Although the queries are not spatially related it has been considered important to see how do the libraries behave when querying the attribute table of a spatial file.

No.	Query	No. of Results
1	SELECT * FROM neigh. WHERE neighborhood LIKE 'Brooklyn'	51
2	SELECT * FROM nyc_zoning WHERE zones LIKE 'PARK'	1549

Table 4: Shapefile Attribute Queries

Within the two different groups, since the queries are subdivided as well between a smaller data set and a bigger one, both PostGIS and Apache Spark's scalability will be tested. Thus, how do their performance behave when the data sets increase their number of features.

Containment Query: Two queries to return all the points that fall within a dataset of polygons.

No.	Query	No. of Results
1	SELECT * FROM trips_split WHERE trips_split WITHIN neigh.	65151
2	SELECT * FROM trips WHERE trips WITHIN neigh.	11498018

Table 5: Within Queries

Intersection Query: Three intersection queries between different geometries:

No.	Query	No. of Results
1	SELECT * FROM neigh WHERE neigh. intersects nyc_zoning	5861
2	SELECT * FROM streets WHERE streets intersects with neigh.	18527
3	SELECT * FROM streets WHERE streets intersects with nyz_zoning	25958

Table 6: Intersection Queries

KNN Query: Two queries that returns the closest "K" features, 1000 in this particular case, from a given feature, e.g. a point.

No.	Query	No. of Results
1	SELECT 1000 closest points to KnnQueryPoint from trips_split	1000
2	SELECT 1000 closest points to KnnQueryPoint from trips	1000

Table 7: kNN Queries

3.2.2 Transformation

Transform every data set from a Geographical Reference System to a Projected Reference System. For this particular case, since all the datasets were retrieved in WGS84 (EPSG: 4326), and considering their geographical location, they will be projected to NAD83 / New York Long Island (EPSG: 2263).

No.	Query	No. of Results
1	SELECT ST_TRANSFORM(trips_split.geom, 2263) FROM trips_split	66454
2	SELECT ST_TRANSFORM(trips.geom, 2263) FROM trips	12484276
3	SELECT ST_TRANSFORM(neigh.geom, 2263) FROM neigh.	195
4	SELECT ST_TRANSFORM(zones.geom, 2263) FROM zones	4547
5	SELECT ST_TRANSFORM(streets.geom, 2263) FROM streets	230682

Table 8: Coordinate Reference System Transformation

3.2.3 Alterations

Due to the functionality limitations this part of the test will only consist in creating a 100 meter buffer for every data set. This is not a simple query, but an Alteration, which will modify the thematic and/or geometric definition of the original data set.

No.	Query	No. of Results
1	SELECT ST_BUFFER(trips_split.geom, 100) FROM trips_split	66454
2	SELECT ST_BUFFER(trips.geom, 100) FROM trips	12484276
3	SELECT ST_BUFFER(neigh.geom, 100) FROM neigh	195
4	SELECT ST_BUFFER(zones.geom, 100) FROM zones	4547
5	SELECT ST_BUFFER(streets.geom, 100) FROM streets	230682

Table 9: Buffer Creation queries

3.2.4 Performance Benchmark Test Datasets

The data used for the Performance Benchmark Test will be open data. In such way, this test can be reproduced by anyone, if necessary.

Variable Name	**Data Type**	**Description**[a]	**Size**	**Feature Count**	**Data Origin**
Trips	Point	CSV Containing pickup points for taxi and limousines in NYC, January 2015	1.8Gb	12000000	Taxi & Limousine Commission (TLC), NYC.[b]
Trips_split	Point	A slice of the Taxi and Limousine Data set	10Mb	65000	Taxi & Limousine Commission (TLC), NYC
Streets	Line	Public streets compiled from orthoimagery for the State of NY	150Mb	230000	NYC Open Data Portal[c]
Neigh.	Polygon	Neighborhoods of NYC	3.4Mb	192	NYC Open Data Portal
Zones	Polygon	NYC Zoning Districts	6.1Mb	4500	NYC Open Data Portal

Table 10: Performance Test Data sets

[a] Tables' schemas can be found in Appendix A
[b] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
[c] http://www1.nyc.gov/site/planning/data-maps/open-data.page

Unfortunately, due to time constraints and a lack of functionality this research has not conducted a Raster Performance Test.

3.3 Use Case

At Wageningen Environmental Research (Alterra), they are currently working and developing the Agro Data Cube. As its own name indicates this is a cube of data containing yearly information about all the fields in The Netherlands since 2012 until today. Currently, it is stored as a database in PostgreSQL.

In this cube, as can be seen in Figure 10 (in Dutch), one axis corresponds to time (years (Tijd)), another one to space (physical representation (Ruimte)) and the last axis is formed by descriptive layers (Kenmerken) such as, weather (Meteo), growth (NDVI), soil type (Bodem), crop type (Gewas), acreage (Areaal) and elevation (Hoogte).

The AgroDataCube contains a set of more than a hundred tables that serve for a wide range of purposes. Table 11 describes the used datasets for this part of the research.

A few queries came up that were suitable for being tested in Agro Data Cube's own environment and Apache Spark. Common queries involve retrieving information of areas or regions, for a specific time span. It was then decided to analyze and compare the performance with the following query type:

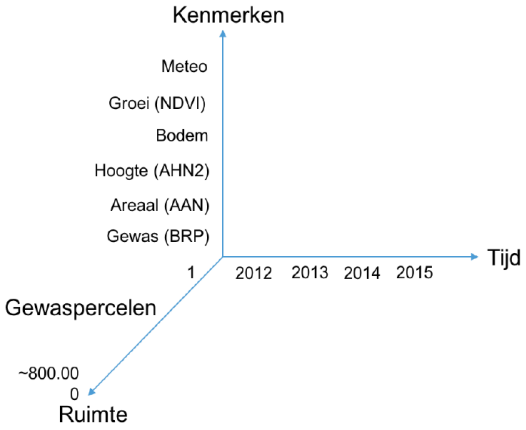


Figure 10: Agrodatacube [Knapen]

- Find all the fields that are within a bounding box and have a certain crop
- Define a point (x,y) and retrieve which crops did it have for all the years
- Define a polygon and find all the polygons that overlap with it throughout the years.
- Define a polygon with a hole in the middle and repeat the above query.
- Overlap the parcels table with the soil physical properties' grid to obtain every field's physical properties.
- Find the best meteorological station per field.
- Retrieve what type of crops did X amount of points had for all the years.

For all the queries that require retrieving information for a point, as this is very small, a fishnet[6] has been created covering the Province of Utrecht, in The Netherlands. The result grid has a 100 meter spatial resolution and contains around 225.000 points. In Table 11 can be seen a description of the different data sets involved.

Variable Name	Data Type	Description[a]	Size	Feature Count
Gewaspercelen	Polygon	Shapefile containing the geometries for all the fields in The Netherlands between 2012 and 2016	3.3Gb	4500000
Meteo_station	Point	Point Dataset containing the different meteorological stations in The Netherlands	1.5 Mb	50
Bofek_2012	Polygon	Vector data set containing the spatial distribution for 72 different physical soil properties in The Netherlands.	2.5 Mb	41251
Utrecht_fishnet	Point	Fishnet of the approximate area of the Province of Utrecht	13.1Mb	225000

Table 11: Use Case Data sets

[a]Table schemas can be found in Appendix A

Although being stored as a traditional RDBMS, the Agro Data Cube is located in a high performance server, an HP Proliant BL460c Gen9 server blade, with 40 Intel Xeon CPU's ES-2650 v3 at 2.3 Ghz, 128 GB RAM

[6]A fishnet is a net of rectangular cells. These cells can be defined as Points, Polylines or Polygons.

memory, and 2 TB storage. Taking into account such environment, this research has decided to find out what would be the specifications for an Apache Spark cluster to perform as fast as the Agro Data Cube server, or how does a cluster with similar specifications as the Agro Data Cube server perform.

4 Functionality Test Results

Next, the results of conducting the functionality benchmark test in the different software packages are presented. The code generated to conduct such tests can be found in https://git.wur.nl/geospatial_spark/functionality_test

4.1 Magellan

4.1.1 Reading Data

Magellan is capable of loading any kind of text file (CSV, TXT, TSV,...), GeoJSON files, Shapefiles and OpenStreetMaps XML files. Well Known Text (WKT) files are stated to be able to be read as well, but there is an implementation missing.

Vector		**Raster**	
File Type	**Supported**	**File Type**	**Supported**
Text Files	1	Text Files	0
XML	1	GeoTIFF	0
WKT	0	Jpeg	0
GeoJSON	1	J2	0
Shapefile	0.8	JSON	0
3.5/10			

Table 12: Magellan Predicates Support

The reason of the grade given to Shapefile in Table 12 is because if the data set contains MultiPolygons it throws an error.

Codewise, loading files is a very straight-forward action. It always follows the same pattern, where the Spark-Context, file path and extra options have to be specified. So, the difference between one file type or another relies in the "type" option specification. Since the library inherits capabilities from Spark-SQL and the data is loaded as a Data Frame, there are plenty of other options that can be specified [Databricks, 2017b].

One of the most useful parameters when loading a text file is that the user can create geometry a column (Point, Line or Polygon) on the fly.

4.1.2 Data types

Magellan has support for the following data types:

- Point
- Line

- PolyLine
- Polygon

Magellan has simple constructor methods to create any single object for the different data types. Point is the most basic feature and with it the rest of object types can be built: e.g. a Line, is defined by two points and a Polyline is nothing but a set of lines. Finally, a Polygon is defined by its vertices, which are Points.

4.1.3 Queries

The fact that it is implemented on top of Apache Spark and that it extends Spark-SQL provides a relational database abstraction for big geospatial analytics [Sriharsha, 2017]. This abstraction implies that coding spatial queries using Magellan does not differ a lot from traditional SQL queries. There are built-in functions to define a bounding box and perform a spatial query on that.

4.1.4 Transformations

Magellan has no implementation yet for any kind of transformation. Neither rasterization nor vectorization are considered, and Coordinate Reference System transformation is not implemented either.

4.1.5 Alterations

Magellan supports the following predicates:

- Within
- Intersects
- Contains

These predicates though, do not apply to all the possible geometry objects combinations. Table 13 illustrates the existing implementations.

	Within	**Intersects**	**Contains**
Point - Line	1	0.5	1
Point - Polygon	1	1	1
Line - Line	0	1	0.5
Line - Polygon	0.25	1	0.5
Polygon - Polygon	0	0.75	0.75
Total	2.25/5	4.25/5	2.75/5

Table 13: Magellan Predicates Support

In Table 13, a Line and a Polyline are treated the same, since they produced the same results. The fact that some predicates do not have a whole point for some pair of data types means that some possible topological relationships have not yet been implemented, e.g. in Line-Polygon the grade is 0.25 because of the four possible topological relations (A Line completely inside a Polygon, a Line crossing a Polygon, a Line in the border of a

Polygon and a Line outside a Polygon), the only implementation that exists is a Line that crosses a Polygon. Similar situations apply for other cases, but for more in depth specification please refer to the repository: https://git.wur.nl/geospatial_spark/functionality_test

Operation			Description	Supported
Reading Data			Accepted file formats	0.35
Queries	Selection	by attributes	Thematic Data Query	1
			Raster Data Query	
	Spatial		Geometric Data Query	1
	Temporal		Temporal Data Query	
Transformations	Coordinate System		Transform from a coordinate system to another	
	Vector -> Raster		Convert a vector file into raster	
	Raster -> Vector		Convert a raster file into vector	
Alterations	Buffering		Create a buffer in a point, line and polygon	
	Intersects		Feature intersects feature	0.8
	Within		Feature within feature	0.5
	Contains		Feature contains Feature	0.6
	Union		Union of two features that share spatial extent	
	Map Algebra	Local	Operations that occur at a cell level	
		Zonal	Operations that occur in a small area around a central cell	
		Focal	Operations that occur in areas that share attributes	
		Global	Operations that occur in the whole raster	
	Distance Operations		Area, length, Euclidean Distance, Cost Distance	
Geostatistics			Spatial interpolation (Kriging) Hot Spot Analysis	
Total				4.5/20

Table 14: Magellan Functionality Test

4.2 GeoPySpark

4.2.1 Reading Data

GeoPySpark is capable of interacting back and forth with a variety of file systems. Any local file system, HDFS (Hadoop Distributed File System), S3 (Amazon web service storage), Apache Cassandra[7], Apache HBase[8] and Apache Accumulo[9].

At the same time, when uploading layers, GeoPySpark stores them in a specific way, known as *catalog*. Figure 11 is an example directory structure of a catalog where every layer of data has its own multilayered data and another layer specifically for the metadata.

Besides being able to interact with many file systems, Table 15 shows the file types GeoPySpark is capable of reading and writing.

There is a unique method, *get*, to load files in memory. But then methods like *write* and *read* can be used to save and retrieve data to and from the catalog. This straight-forward manner of dealing with data set simplifies the data handling process.

```
layer_catalog/
  layer_a/
    metadata_for_layer_a/
      metadata_layer_a_zoom_0.json
      ....
    data_for_layer_a/
      0/
        data
        ...
      1/
        data
        ...
      ...
  layer_b/
  ...
```

Figure 11: Example of a GeoPySpark catalog [GeoPySpark, 2018]

Vector		Raster	
File Type	**Supported**	**File Type**	**Supported**
Text Files	0	Text Files	1
XML	0	GeoTIFF	1
WKT	0	Jpeg	1
GeoJSON	0	J2	1
Shapefile	0	JSON	1
Total: 5/10			

Table 15: GeoPySpark File Types

4.2.2 Queries

There are two ways of retrieving data from a saved layer in the catalog. One is reading the entire layer using the method *read*, and the other is selecting a portion using the method *query*.

When using the method *query* a geometry can be passed to clip the specific area of interest. This geometry can be passed as a *shapely.geometry*[10] (which can be either a Polygon or a MultyPolygon), the *WKB* representation of the geometry, or an *Extent*. This will define one or more windows and the pixels that fall completely inside it will be returned.

[7] Apache Cassandra is a NoSQL database management system.
[8] Apache HBase is a distributed, non-relational database.
[9] Apache Accumulo is a sorted, distributed key/value store.
[10] Shapely is a Python library dedicated to the manipulation and analysis of geometric objects in the Cartesian plane [Shapely, 2017].

By default, the same Coordinate Reference System is expected in both the geometry and the queried layer, but if that is not the case, the EPSG code can be specified while querying.

In addition, GeoPySpark allows the creation of Spatial Temporal layers. This means that every layer can have one or more time-stamp column which will allow temporal queries. The layer can then be queried by a specific instant or an interval.

The user can then decide to query by extent, time or both at the same time by just specifying or not the desired parameters when typing the query

4.2.3 Transformations

GeoPySpark has bindings for *Proj.4* [OSGeo, 2016], which is a standard UNIX filter function that performs conversions between cartographic projections. This allows to transform or reproject any uploaded layer.

Finally, since it is a raster library, there is a *Rasterize* method. It determines the set of pixel values covered by each vector element and assigns a supplied value to that set of pixels in a target raster. One inconvenient of this method is the fact that the geometries must be *Shapely* geometries and there is no specification on the possible resampling methods.

4.2.4 Alterations

GeoPySpark has support for Map Algebra operations on rasters. Unfortunately, as of today and unlike its parent GeoTrellis, GeoPySpark only allows *local* and *focal* operations.

Those operations run only on *TiledRasterLayer* and if a local operation requires multiple inputs, those inputs must have the same layout and projection.

Local operations include arithmetic operations (addition, substraction, multiplication and division) using *integers*, *floats* or other *TiledRasterLayer*.

Focal operations need an operation and a neighborhood definition. The neighborhood can be a square (squared neighborhood), a circle (the radius determines which cells fall within the bounding box), a wedge (a wedge neighborhood; a radius, a starting angle and an ending angle have to bee specified), an annulus (ring) and a NESW (North, East, South, West).

Finally, cost distance operations can also be performed using GeoPySpark.

Operation			Description	Supported
Reading Data			Accepted file formats	0.5
Queries	Selection	by attributes	Thematic Data Query	1
			Raster Data Query	1
	Spatial		Geometric Data Query	1
	Temporal		Temporal Data Query	1
Transformations	Coordinate System		Transform from a coordinate system to another	1
	Vector -> Raster		Convert a vector file into raster	1
	Raster -> Vector		Convert a raster file into vector	
Alterations	Buffering		Create a buffer in a point, line and polygon	
	Intersects		Feature intersects feature	
	Within		Feature within feature	
	Contains		Feature contains Feature	
	Union		Union of two features that share spatial extent	
	Map Algebra	Local	Operations that occur at a cell level	1
		Zonal	Operations that occur in a small area around a central cell	1
		Focal	Operations that occur in areas that share attributes	
		Global	Operations that occur in the whole raster	
	Distance Operations		Area, length, Euclidean Distance, Cost Distance	1
Geostatistics			Spatial interpolation (Kriging) Hot Spot Analysis	
Total				9.5/20

Table 16: GeoPySpark functional test

4.3 GeoSpark

4.3.1 Reading Data

Table 17 shows the fie types that GeoSpark is capable of loading.

Vector		Raster	
File Type	**Supported**	**File Type**	**Supported**
Text Files	1	Text Files	0
XML	0	GeoTIFF	0
WKT	1	Jpeg	0
GeoJSON	1	J2	0
Shapefile	1	JSON	0
Total 4/10			

Table 17: GeoSpark Reading capabilities

Besides file types that appear in Table 17, GeoSpark has support for NASA Earth Data NetCDF/HDF[11]

For CSV, TSV, WKT and GeoJSON it is enough with specifying the *FileDataSplitter* when using the constructor to load the data set. Every accepted Data Type (Circles, LineStrings, Points, Polygons or Rectangles) has a variety of different constructors, depending on the arguments given, but all of them need the location ad the splitter type, at least. For both Esri Shapefile and NASA Earth Data NetCDF/HDF there are specific methods.

4.3.2 Queries

After geometrical objects are retrieved as Spatial RDD or Spatial Data Frames, users can then call any spatial query operations defined. GeoSpark will run over the in-memory cluster, decide how spatial object-relational tuples could be stored, indexed and accessed using SRDDs. It then returns the spatial query results. The built-in queries shown below:

- **Range Query**: In GeoSpark, range query is constrained to a defined query window in the form of an *Envelope*.
- **Join Query**: There are two different types of Join Queries:
  - **Distance Join Query**: Inner joins two sets of geometries on a given distance from one of them. What is commonly known as a buffer. It returns RDD of pairs where each pair contains a geometry and a set of matching geometries.
  - **Spatial Join Query**: Inner joins two sets of geometries on "Contains" or "Intersects" relationship, which has to be specified when calling the method. It returns RDD pairs, where each pair contains a geometry and a set of matching geometries.
- **KNN Query**: Returns the "K" closest features from a layer to a point.

[11]HDF: Hierarchical Data Format is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data. NetCDF (network Common Data Form) is a data model for array-oriented scientific data, as well as a freely distributed collection of access libraries that support implementation of the same data model, and a machine-independent data format. Together, the interfaces, libraries, and format support the creation, access, and sharing of scientific data.

4.3.3 Transformations

GeoSpark has support for Coordinate Reference System transformations, by means of supplying the EPSG code, of any kind of data type. Once the method is called, for the transformation to persist, *Analyze* method must be applied to the desired variable. There is no support for any transformation between raster and vector or vice versa.

4.3.4 Alterations

Finally, in GeoSpark has a set of Geometrical Operations defined, as can bee seen in Table 18. Besides the three predicates shown, **Polygon Union** and **Boundary** predicates can also be performed

	Within	Intersects	Contains
Point - Line	1	1	1
Point - Polygon	1	1	1
Line - Line	1	1	1
Line - Polygon	1	1	1
Polygon - Polygon	1	1	1
Total	5/5	5/5	5/5

Table 18: GeoSpark Predicates support

Operation			Description	Supported
Reading Data			Accepted file formats	0.5
Queries	Selection	by attributes	Thematic Data Query	1
			Raster Data Query	
	Spatial		Geometric Data Query	1
	Temporal		Temporal Data Query	
Transformations	Coordinate System		Transform from a coordinate system to another	1
	Vector -> Raster		Convert a vector file into raster	
	Raster -> Vector		Convert a raster file into vector	
Alterations	Buffering		Create a buffer in a point, line and polygon	1
	Intersects		Feature intersects feature	1
	Within		Feature within a feature	1
	Contains		Feature contains Feature	1
	Union		Union of two features that share spatial extent	1
	Map Algebra	Local	Operations that occur at a cell level	0
		Zonal	Operations that occur in a small area around a central cell	0
		Focal	Operations that occur in areas that share attributes	0
		Global	Operations that occur in the whole raster	0
	Distance Operations		Area, length, Euclidean Distance, Cost Distance	
Geostatistics			Spatial interpolation (Kriging) Hot Spot Analysis	
Total				8.5/20

Table 19: GeoSpark functional test

4.4 GeoMesa

Unfortunately it has not been possible to realize any of the test above mentioned to GeoMesa. Its complicated architecture and its focus towards a more software engineering audience made impossible for this research to even set it up due to time constraints.

5 Results on performance test

In order to conduct the performance test in a cluster environment, the Amazon Web Services EC2 [Amazon Web Services, 2017a] have been chosen. At the same time, benefiting from the AWS Educate Starter Account [Amazon Web Services, 2017c], the m4.large instance type [Amazon Web Services, 2017b] was used. This type of instances have two CPU's each, with 6Gb memory and a dedicated bandwidth of 450 Mbps. In addition, Flintrock [Chammas, 2018], a command-line tool for launching Apache Spark test clusters has been used. In such a way, for every test 5 instances were launched, 4 slaves and 1 master, with 6.0 Gb of dedicated memory.

The queries tested in PostGIS where launched in an Asus laptop with an Intel Core i7 processor and 8 Gb of RAM.

The code generated for this test can be found in https://git.wur.nl/geospatial_spark/performance_test

Every test has been executed a minimum of ten times and the average time is presented, as well as the standard deviation. Five different times have been measured:

1. **Loading times**: The time every library needed to load the necessary data sets.
2. **Warm-up times**: This is the measured time for the first query done to a data set. Since the first query is always 10x to 20x times slower than the rest it has not been taken into account in the average but has been measured separately.
3. **Transformation**: The time every library needs to transform from one CRS to another.
4. **Query times**: The average result of running ten times the defined queries.
5. **Feature creation**: Creating a buffered feature for every data set

Following, the time results of applying the performance test on the libraries are shown.

5.1 Loading Times & Warm Up Times

Although during the entire data analytics process it may happen only once, if no mistakes are made, loading the data sets into the database or the desired environment can take plenty of time. This is why this time has been measured (Table 20).

	Size	**PostGIS**	**Magellan**	**GeoSpark**
Trips (CSV)	2.2 Gb	1200 (1.3)	18.93 (0.16)	20.18 (0.99)
Trips_split table (CSV)	11.6 Mb	120 (1.2)	10.60 (0.26)	1.25 (0.06)
Neighborhoods (GeoJson)	1.7 Mb	0.5 (0.01)	0.62 (0.01)	0.69 (0.015)
NYC Zones (Shapefile)	6.4 Mb	1.0 (0.02)	1.25 (0.33)	1.07 (0.12)
NY State (Streets)	146 Mb	22.35 (1.29)	1.38 (0.28)	1.015 (0.08)

Table 20: Loading Data Sets Times (s), 10 repetitions

Secondly, the first query time has not been considered into the final average calculation, but has been measured as well as seen in Table 21.

	Size	**PostGIS**	**Magellan**	**GeoSpark**
Trips (CSV)	2.2 Gb	28 (0.05)	24.29 (0.41)	12.15 (0.56)
Trips_split table (CSV)	11.6 Mb	0.52 (0.02)	3.33 (0.26)	1.49 (0.25)
Neighborhoods (GeoJson)	1.7 Mb	0.15 (0.01)	1.05 (0.06)	2.28 (0.17)
NYC Zones (Shapefile)	6.4 Mb	0.085 (0.0)	1.89 (0.12)	0.89 (0.08)
NY State (Streets)	146 Mb	0.15 (0.05)	18.62 (1.23)	20.50 (0.81)

Table 21: Warm-up Query Times (s), 10 repetitions

5.2 Attribute Queries

5.2.1 Queries to CSV's

In Figure 12, queries 1 and 2 have been performed to the Trips_split CSV file with 65000 records and queries 3 and 4 have been performed to the Trips CSV file with 12000000 records. It can be seen that those libraries that benefit from Apache Spark do not scale-up, whilst in PostGIS, for a dataset 180 bigger it performs 30 times slower.

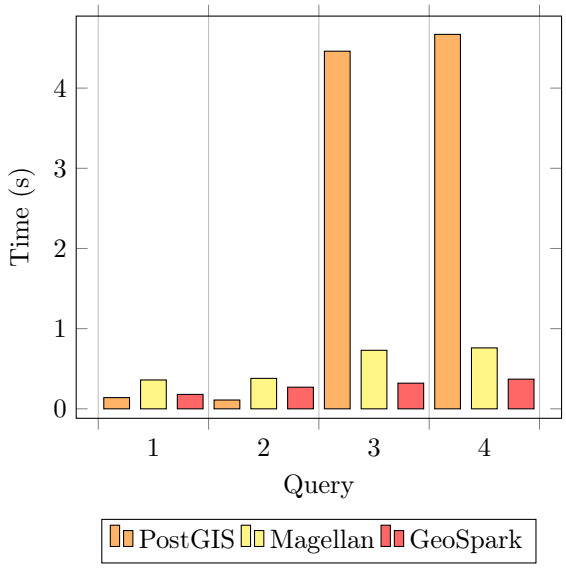


Figure 12: CSV Attribute Query Times (s), 10 repetitions

No.	**PostGIS**	**Magellan**	**GeoSpark**
1	0.14 (0.01)	0.27 (0.03)	0.18 (0.01)
2	0.11 (0.01)	0.34 (0.04)	0.27 (0.03)
3	4.46 (0.02)	0.78 (0.24)	0.69(0.05)
4	4.67 (0.02)	0.83 (0.10)	0.57 (0.07)

Table 22: CSV Attribute Query Times (s), 10 repetitions

In Figure 13, the first query was performed to the Neighborhoods dataset containing 192 polygons and the second query was performed to the Zones dataset containing 4500 polygons. It can be seen that those libraries that benefit from Apache Spark do not scale-up, whilst in PostGIS, for a data set 23 times bigger, it performs 5 times slower.

5.2.2 Queries to Shapefiles

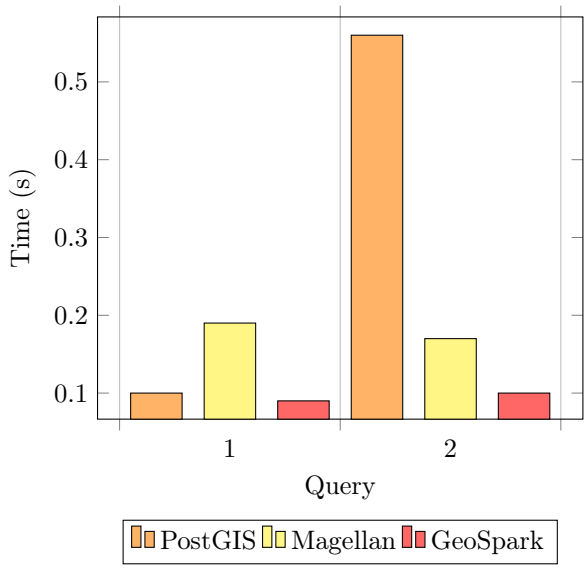


Figure 13: Shapefile Attribute Query Times (s), 10 repetitions

No.	**PostGIS**	**Magellan**	**GeoSpark**
1	0.10 (0.01)	0.19 (0.03)	0.09 (0.01)
2	0.56 (0.02)	0.17 (0.02)	0.1 (0.02)

Table 23: Shapefile Attribute Query Times (s), 10 repetitions

5.3 Intersection Queries

Three intersection queries have been performed.

1. Neighborhoods (Polygon, 192 features) - Zones (Polygon, 4500 features)
2. Streets (Line, 230000 features) - Neighborhoods (Polygon, 192 features)
3. Streets (Line, 230000 features) - Zones (Polygon, 4500 features)

As it can be depicted from Figure 14, the first thing that stands out visually is the amount of time that it takes to Magellan to perform these queries.

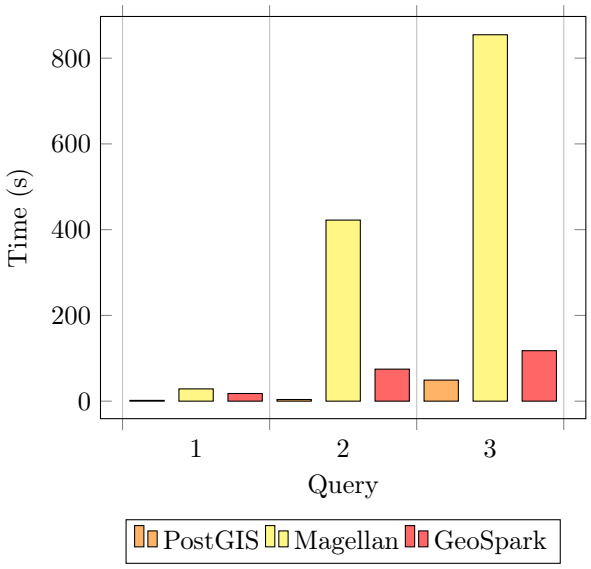


Figure 14: Intersection Query Times (s), 10 repetitions

PostGIS	Magellan	GeoSpark
1.82 (0.73)	28.62 (1.13)	15.52 (0.59)
3.835 (0.85)	422.32 (5.45)	54.16 (1.52)
49.1 (1.29)	854.58 (8.34)	117.67 (1.80)

Table 24: Intersection Query Times (s), 10 repetitions

5.4 Within Queries

Two within queries have been performed.

1. Trips_split (Point, 65000 features) Within Neighborhoods (Polygon, 192 features).
2. Trips (Point, 12000000 features) Within Neighborhoods (Polygon, 192 features).

For this type of query, the library whose results stand out is GeoSpark, note the logarithmic y-axis scale in Figure 15. For an increase of 180 times more features, Magellan performs 33 times slower, PostGIS 411 times slower and GeoSpark could not be measured.

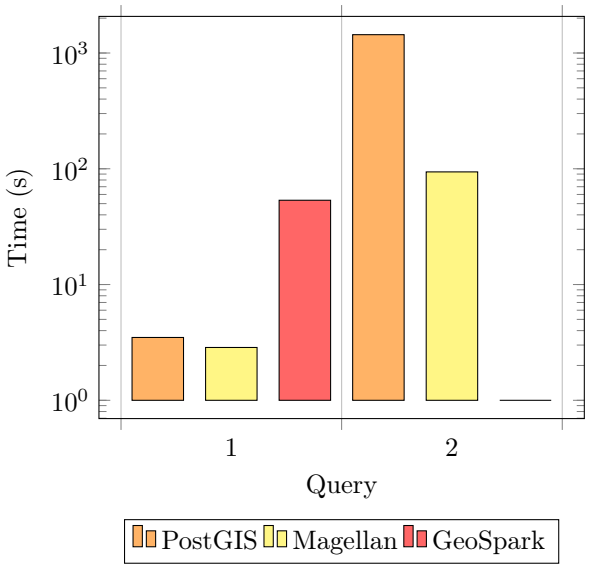


Figure 15: Within Query Times (s), 10 repetitions

PostGIS	Magellan/	GeoSpark
3.49 (0.74)	2.86 (0.14)	53.47 (1.18)
1440 (3.29)	93.98 (7.11)	NULL

Table 25: Within Query Times (s), 10 repetitions

5.5 kNN Query

Two kNN queries have been performed:

- Return the 1000 closest points to a given point from Trips_split (Point, 65000 features).
- Return the 1000 closest points to a given point from Trips (12000000 features).

As it can be seen in Figure 16, PostGIS query times increase around 180 times when the dataset is also 180 times bigger, whilst in GeoSpark the increase is only of around 25 times.

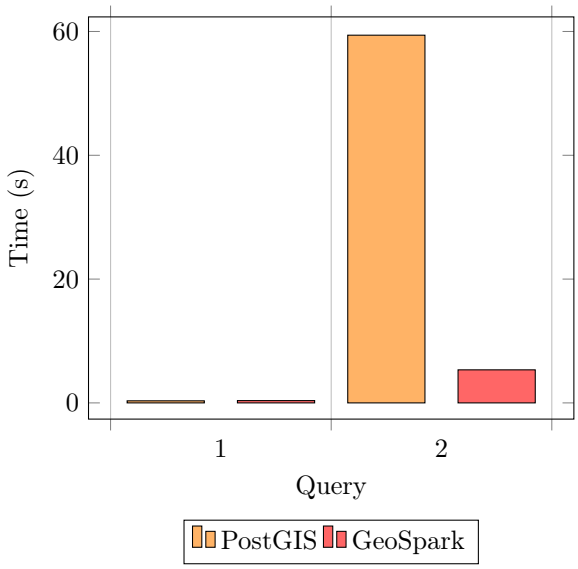


Figure 16: kNN Query Times (s), 10 repetitions

PostGIS	GeoSpark
0.336 (0.05)	0.372 (0.03)
59.4 (1.23)	5.34 (0.38)

Table 26: kNN Query Times (s), 10 repetitions

5.6 Transformation

What can be clearly depicted from Figure 17 is that PostGIS increases exponentially its performance time as the data sets increase in size. Transforming the Trips data set (12000000 features) took more than one hour, compared to GeoSpark wich took a bit more than a minute.

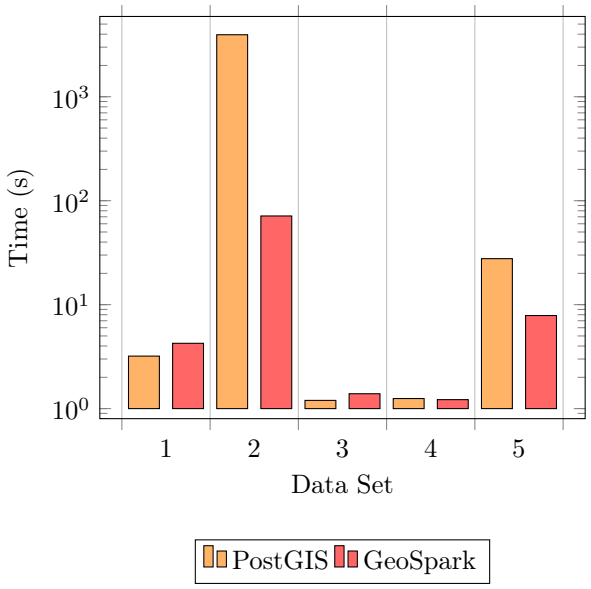


Figure 17: Transformation Times (s), 10 repetitions

No.	Dataset	**PostGIS**	**GeoSpark**
1	Trips_split	3.2 (0.02)	4.25 (0.89)
2	Trips	3950 (2.31)	71.32 (1.25)
3	Neighborhoods	0.70 (0.01)	0.86 (0.02)
4	Zones	0.71 (0.02)	0.72 (0.03)
5	Streets	27.7 (0.09)	7.86 (0.05)

Table 27: Transformation Times (s), 10 repetitions

5.7 Alterations

What can be depicted from Figure 18 is that PostGIS increases exponentially its performance time as the data sets increase in size. Both creating a buffer around the 250000 lines data set and 12000000 point data set took more than one hour, compared to GeoSpark, which took than 15 seconds in both cases.

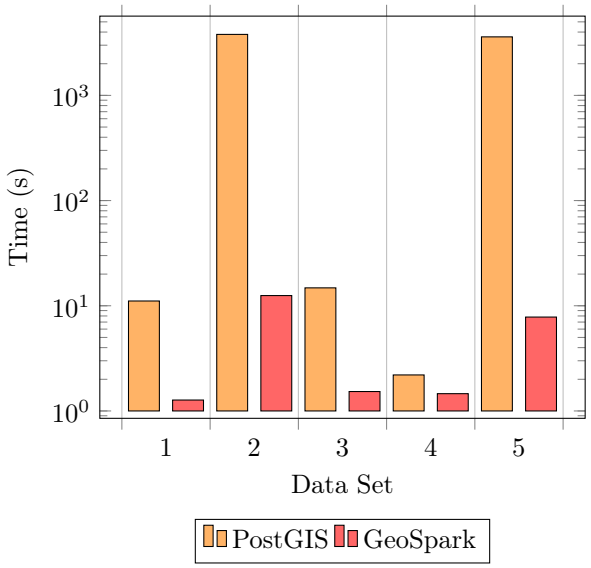


Figure 18: Feature Creation times (s), 10 repetitions

No.	Dataset	**PostGIS**	**GeoSpark**
1	Trips_split	11.1 (0.05)	0.27 (0.06)
2	Trips	3800 (10.23)	12.50 (0.54)
3	Neighborhoods	14.8 (0.93)	0.53 (0.04)
4	Zones	2.2 (0.89)	0.46 (0.05)
5	Streets	3600 (15.58)	7.81 (0.53)

Table 28: Buffer Creation Times (s), 10 repetitions

Finally, with the aim of testing Apache Spark's horizontal scalability, as well as seeing the performance time change with more computing power, the same performance test was conducted both on a set of 4 and 8 nodes.

Table 29 shows the results for GeoSpark and Table 30 shows the results for Magellan.

		4 nodes		8 nodes	
	Dataset \ Query num.	Mean (s)	St. Dev.	Mean (s)	St. Dev.
Loading times	Trips Split	1.34	0.15	1.25	0.06
	Big Trips	32.11	1.06	20.18	0.99
	Neighborhoods	0.69	0.12	0.69	0.02
	Zones	0.03	0.03	0.02	0.004
	Streets	0.02	0.001	0.002	0.005
Warming Up times	Trips Split	0.65	0.03	0.68	0.003
	Big Trips	0.77	0.17	0.87	0.21
	Neighborhoods	2.43	0.18	2.47	0.04
	Zones	2.57	0.968	3.23	0.19
	Streets	21.27	0.952	20.51	0.08
Attribute Query CSV	1	0.1	0.01	0.18	0.01
	2	0.28	0.05	0.27	0.02
	3	0.34	0.07	0.32	0.02
	4	0.68	0.15	0.37	0.23
Attribute Query Shapefile	1	0.09	0.01	0.1	0.02
	2	0.08	0.00	0.1	0.01
Intersects Query	1	15.05	0.82	15.52	0.33
	2	56.97	2.37	54.16	1.52
	3	112.94	4.49	117.67	1.80
Within Query	1	57.38	2.97	51.91	1.18
	2	NaN	NaN	NaN	NaN
Buffer Query	1	4.02	0.45	4.09	0.17
	2	10.49	0.48	10.21	0.54
KNN Query	1	0.45	0.06	0.36	0.03
	2	8.88	0.07	5.34	0.38

Table 29: GeoSpark 4 nodes vs. 8 nodes

	Dataset \ Query num.	4 nodes		8 nodes	
		Mean (s)	St. Dev.	Mean (s)	St. Dev.
Loading times	Trips Split	11.40	2.37	10.60	0.5
	Big Trips	28.69	0.57	18.93	0.16
	Neighborhoods	0.55	0.06	0.6	0.01
	Zones	0.28	0.04	0.31	0.06
	Streets	0.44	0.06	0.48	0.01
Warming Up times	Trips Split	3.09	0.19	3.33	0.26
	Big Trips	42.81	0.75	24.29	0.41
	Neighborhoods	1.01	0.09	1.05	0.06
	Zones	1.45	0.24	1.89	0.12
	Streets	20.82	2.02	18.71	1.23
Attribute Query CSV	1	0.38	0.03	0.27	0.03
	2	0.34	0.05	0.36	0.04
	3	0.58	0.16	0.73	0.24
	4	0.63	0.16	0.84	0.10
Attribute Query Shapefile	1	0.19	0.03	0.17	0.03
	2	1.73	0.20	0.17	0.02
Intersects Query	1	28.62	1.11	28.7	0.54
	2	422.32	5.35	415.97	1.75
	3	854.58	8.34	827.92	0.94
Within Query	1	2.69	0.14	2.85	0.23
	2	168.83	7.11	93.98	1.22

Table 30: Magellan 4 nodes vs. 8 nodes

6 Results on AgroDataCube

Due to time constraints, only one query was finally put into practice. This is retrieving the crops for all the years for a fishnet of points covering the Province of Utrecht. This single query though, was too big for a small sized cluster, such as the ones used for the Performance Benchmark test, is no longer enough. This is why and, in order to emulate Agro Data Cube server's performance, the same query, for different sized data sets, have been run in 4, 8 and 16 nodes clusters and in two different types of clusters:

- m4.large: 2 vCPU, 8 Gb memory and 450 Mbps Dedicated EBS Bandwidth.
- m4.xlarge: 4 vCPU, 16 Gb memory and 750 Mbps Dedicated EBS Bandwidth.

Both type of clusters are eligible for the Amazon Web Services Educate Starter Account.

Following, Table 31 shows the obtained results during the tests.

	m4.large			m4.xlarge			PostGIS
Points	**4 Nodes**	**8 Nodes**	**16 Nodes**	**4 Nodes**	**8 Nodes**	**16 Nodes**	–
10^3	1.3	0.81	0.5	0.83	0.45	0.25	0.52
10^4	17	7.4	4.1	7.45	4.02	1.96	0.75
10^5	–	76.15	41.93	–	41.05	19.89	1.85
250215 (all)	–	–	141.82	–	–	75.96	5.7

Table 31: Point Within Polygon Times (min)

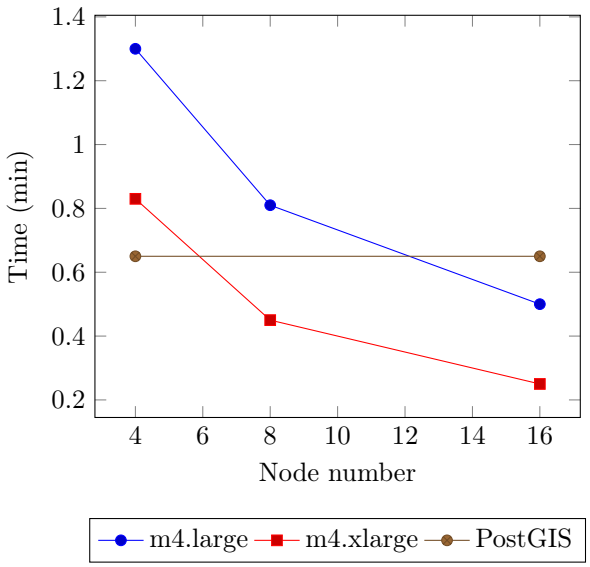


Figure 19: 10^3 points query times (min)

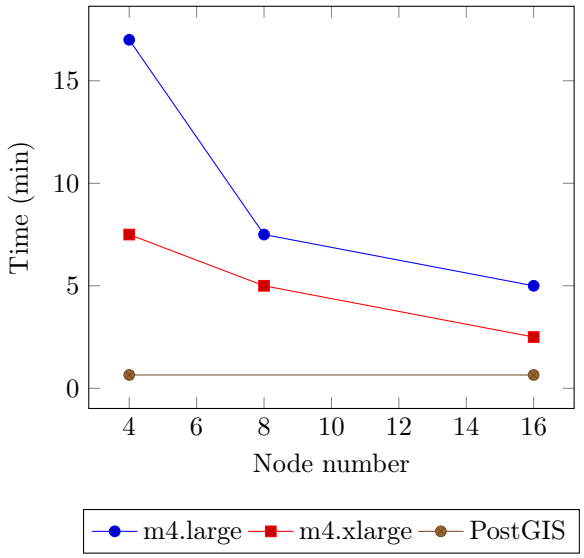


Figure 20: 10^4 points query times (min)

Figures 19 and 20 show the times of performing the query to a subset of 1000 and 10000 points, respectively. It also shows the time taken to perform the same query using PostGIS in a regular computer.

7 Discussion

From the results obtained during the functionality test, it can be stated that the existing functionality in the tested libraries is not yet developed enough to cope with the needs of GIS. Not solely the lack of operations, but also the lack of some topological relations, describes a very low development in functionality still. At the same time, the final grade in the test should be taken carefully into account. All the libraries are either dedicated to Raster or to Vector. This is one of the reasons of the low grades all libraries obtained in the functionality tests.

The functionality analysis showed as well that there are two different trends for solving the problems that arise when dealing with GeoSpatial Data in parallel distributed systems. The first one observed, in Magellan, is to make use of all the potential of SparkSQL and store the data in extended Data Frames with a spatial component. A (Spatial) Data Frame is not different from a table in a RDBMS. They can be queried, joined or sliced as regular tables, because all the underlying parallel processes are not seen by the user.

The latter, as seen in GeoPySpark and GeoSpark is to make use of the existing Spark Data types, RDD's (Resilient Distributed Data Sets). All data that is dealt with using GeoPySpark is, at some point of the process, stored within a RDD. GeoPySpark does not work with PySpark RDDs directly, but rather uses Python classes that are wrappers of classes in Scala that contain and work with a Scala RDD. GeoSpark, on the other hand, has a super class named *SpatialRDD* and any type of geometry is a subclass of it (PointRDD, RectangleRDD, CircleRDD, LineStringRDD and PolygonRDD). Starting from this, every variable can then be converted to a Data Frame if needed.

Regarding the performance test there are a couple results that stand out. To begin with, there are two queries that run strangely slow in Apache Spark. These are the Intersection Query for Magellan and the Within Query for GeoSpark.

In the former this is due to a miss-functionality in indexing line data sets, which involve the second and third queries [Sriharsha, 2018 (Personal communication)]. Without this implementation, it does not matter whether the calculation is parallelized or not, because they are highly time demanding. Beyond this noteworthy result, it is important to mention that in no case has any Apache Spark library performed faster than PostGIS regarding the intersection queries. This could be due to different reasons, but it is likely that either the data sets are not big enough to push PostGIS to its limits and prove the benefits of parallelizing, or that the algorithms behind PostGIS are far better than those algorithms in Magellan and GeoSpark.

For the latter, the reason behind this performance is due to the fact that GeoSpark uses JTS[12] library to do 1:1 point-polygon test in parallel, which is the case for both Within queries. However, JTS point-polygon query is much slower than some existing libraries, such as ESRI geometry API. From GeoSpark they are trying to replace JTS with other libraries, or overwrite the existing JTS alrgorithms (which is being done in many cases), but as many open-source projects, there is a lack of time to implement such improvements [Yu, 2018 (Personal communication)]. In the case of Magellan, this join is executed as a Broadcast Nested Loop Join [Sriharsha, 2017]. Magellan performed up to 15 times faster than PostGIS for the second query.

Because of these two performance problems in can not be depicted which of the two approaches (Spatial Data Frames or Spatial RDD's) performs better when dealing with Geospatial data.

Parallelizing showed improvement again in the kNN query, because this type of queries are very compute-intense. When the data size is 180 times bigger, PostGIS performs almost 180 times slower whilst GeoSpark only around 20 times slower.

Regarding both Coordinate Reference System Transformation and feature creation, it can be seen the exponential increase in time, as the data set size increases, in PostGIS. It could be arguable the fact that this was run not in a very powerful computer, but the interesting aspect is the underlying notion that while using a traditional RDBMS performance time increases exponentially, whilst in a parallel distributed system it increases lineally.

[12]The JTS Topology Suite is a Java library for creating and manipulating vector geometry. It also provides a comprehensive set of geometry test cases, and the TestBuilder GUI application for working with and visualizing geometry and JTS functions. [LocationTech]

Another result that backs the data sets size statement is what can be seen in Tables 29 and 30. Performance times do not improve significantly or at all from a 4 node cluster to an 8 one. There are only two specific cases that, when doubling the number of clusters, the performance time reduces by nearly a half; the Intersection Query in Magellan and the KNN query in GeoSpark. The first one is because kNN queries require that the entire table is read before sorting out [Huang et al., 2017a], which is very computing demanding. In the Intersection Query this happens because, as Magellan has no correct indexing line data sets, the query is not optimized and, by adding more executers (so computing power) the process speeds-up.

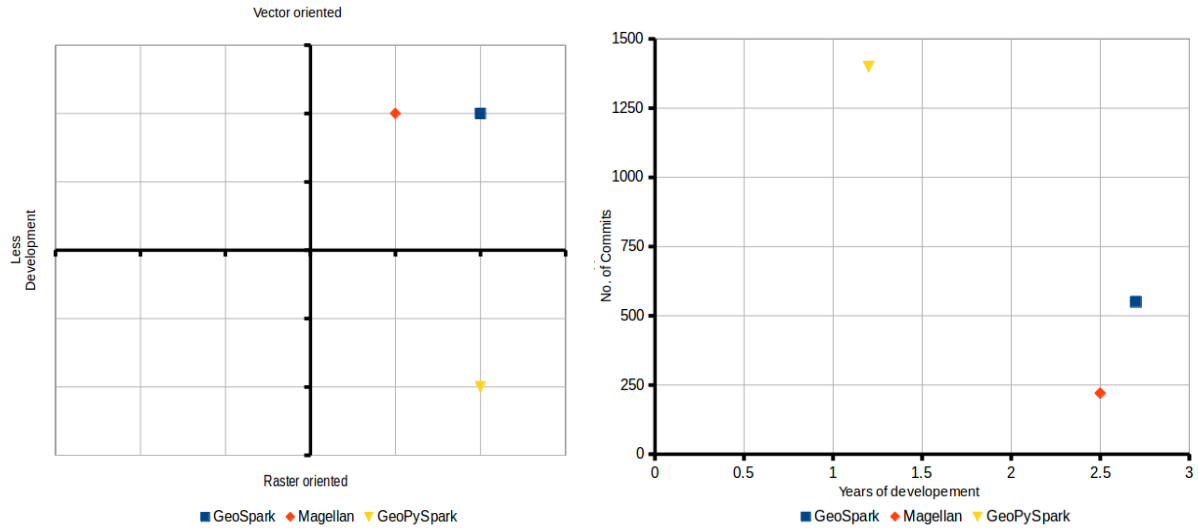
Finally, the software packages performance was applied to a use case. Although this was successfully designed, the time constraint has not permitted to develop further analysis on the results that are not quite clear, i.e results on Table 31 show that in this case PostGIS performed up to 40 times faster than Magellan. Although the query (within) was also used in the performance test, see Table 25 and Figure 15 where Magellan performed up to 30 times faster than PostGIS, the datasets in this case are different. One of the reasons for the performance evolution could be the polygons' granularity. Whilst in the performance test the geometries were the different neighborhoods of new York city, in the Use Case those polygons are the different fields in The Netherlands. For the first case, the geometries are more complex than in the second case. It appears that PostGIS can really boost its performance when the geometries are simpler [Santos et al., 2015]. Magellan on the other hand does not seem to make any difference in performance no matter the complexity of the polygons.

It could be arguable that the chosen query was not the most suitable to test in Apache Spark, but it is what came out for the use case. Nevertheless this last step of the research served the purpose of testing Apache Spark's horizontal scalability. As Table 31 and Figures 19 and 20 show, although being slower than PostGIS, performance times decrease when more clusters are added. Despite of being logical and expected, this proves that without any change in the code, just by deploying it to more nodes, the performance increases whereas the results obtained from the PostgreSQL/PostGIS data base can not be improved once the physical limits are reached. Besides parallelizing, Apache Spark also provides robustness. This allows for a continuous query utility as long as half of the nodes within the cluster are alive [Wang and Khan, 2015]. In comparison with a traditional RDBMS where only one master-slave mode is ensured [Sanders and Shin, 2001].

During this step of the research, the tuning possibilities Apache Spark provides have been tested. When launching an operation in a cluster, in order to maximize the existing resources, it is crucial that all nodes are busy at all time. So, Apache Spark has a default parallelization, but it can be changed for every specific need.

As an example, when running the first queries in a m4.large cluster, with 8 nodes (16 executors), it was noted that Apache Spark divided the operation in 24 partitions. This meant that, during the first go, 16 executors were busy, but only 8 were operating during the second round. When that was tuned to 32 partitions, which also meant that every partition was a bit smaller in memory, performance time decreased my almost 25%. This proves that the potential of Apache Spark does not end in launching operations but its complete flexibility to adapt to every particular situation.

Although the results do not show a great level of maturity, there are indicators that leave room for hope. In Figure 21a the relative development of the three software packages tested is shown as well as which data type are they more focused on. As it can be seen, GeoPySpark and GeoSpark are the most developed. Figure 21b also shows this by presenting the number of commits and the years that the software packages have been under development. It can be seen that GeoPySpark, being the youngest, has more than two times commits than the other software packages.



(a) (b)

Figure 21: Data type vs. Development (a) and Years of development vs. Number of commits (b)

8 Conclusions

There are many ways of creating a Functionality Benchmark Test for GeoSpatial Big Data, depending on what its purpose would be. Since one of the goals of this research was to define what are the most basic and necessary tools or operations within GIS, the literature review was based upon that. Then, once these operations are defined, it is important to define such test in an incremental way. From "simpler" operations to more "complex" ones. At the same time, such test must be done extensively, testing every predicate for every data pair possible. It can be concluded then that, although there are many ways of creating a Functionality Benchmark Test, it would be desirable to develop a standardized test in order to have a rule on which base future software packages classification.

After defining and applying the benchmark tests it can be concluded that these libraries do not have yet the necessary functionality to be used for a more extensive use within the Geo Information Science scope. As of today, the early stages of development, added to the fact that the vast majority of the geospatial scientific community is unaware of Apache Spark, leads to the undeniable situation where a complete spin towards the parallel paradigm is not yet possible.

Results and current development show that, despite the actual lack of functionality, it is a matter of time for it to become a reality. This can perfectly be seen in Figures 21a and 21b, which show both the youth of these software packages and the relative speed of development. As mentioned before in this report, once the data sets become too big to be dealt with in normal computers or with traditional software there will always be the need to change paradigms.

Even though their functionality development level does not allow a more extended use, this research proves that the base on which to build further development exists, i.e. it is possible to analyze geospatial data in a parallel distributed environment. Furthermore, one of the greatest things about developing on top of Apache Spark and also the base of this research is the fact that a Apache Spark based Spatial Database can be "endlessly" extended horizontally. This is, increasing the number of backend servers.

The tested software packages have performed better than an RDMBS tool in operations that were computing demanding, such as the kNN query, Coordinate Reference System Transformation or Creating new features, like the Buffer case. On the other hand, in spatial queries like Intersect or Within (during the Use Case), PostGIS performed better than GeoSpark or Magellan. This proves that, nowadays, it only makes sense to

perform GeoSpatial Analytics when the analysis itself is of a high computational demand. Even though PostGIS performs in a sequential manner (compared to the parallel approach of Apache Spark), its algorithms are very efficient and it is a powerful tool for Geospatial analytics.

These software packages can replicate the results obtained with traditional GIS software in a production environment, as long as the predicates that are needed for the algorithm have been defined. As mentioned previously, Apache Spark can be extended horizontally as much as needed, so its performance could ultimately always reach the one of any high performance server. Furthermore, Apache Spark allows to be tuned while launching jobs in a cluster, which makes it a highly flexible tool capable of adapting to every kind of situation. This means that one could analyze every step or operation needed and change Apache Spark's inner parameters to perform to its best with the available resources.

Finally, from the self experience of conducting this research, there are a couple of things that deserve to be mentioned. First, the fact that these software packages are developed by teams that are not necessarily GIS developers creates confusion when it comes to naming conventions, for example. Secondly, although being in a very experimental state in most of the cases, the easiness in which the developers could be reached made it possible for this research to reach its actual point. Lastly, despite not reaching the desired performance in the last step of the research, this has proven that by means of Apache Spark and a simple laptop anyone with a small knowledge of software can set up and environment to process Big GeoSpatial Data.

9 Recommendations

This research serves as a starting point for further research based on Geospatial Analysis in Apache Spark. Due to time constraints it has only been possible to develop a Vector Performance Test, but it would be very important as well to develop and perform one for Raster Data. At the same time, it would be of very importance to further investigate in the results obtained during the Use Case application as well as to try new applications or scenarios that could make use of big geospatial analysis tools.

10 Acknowledgements

This research has been probably the most intense 6 months of my life. Obviously, I have not been alone during the entire process, so I would like to thank to everyone involved, direct or indirectly, in a successful research.

Ioannis and Arend, thank you for your support, experience and tips during the entire process. This would have not been possible without your help and passion.

Rob Knapen, from Alterra, without your interest and patience this research would be definitely incomplete.

Jia Yu, from GeoSpark, Jacob Bouffard from GeoPySpark and Christos Charmatzis and Ram Sriharsha from Magellan, without your help in setting up and understanding your creations, this thesis would have never gotten that far.

Xènia, gràcies per confiar en mi a totes hores i en tot moment. Per aguantar quan estic decaigut i fer-me riure i feliç sempre que et tinc a prop. Gràcies per creuar-te a la meva vida.

Mama, gracias por estar siempre allí y escucharme cuando lo necesito. Por ser ese pilar en el que puedo confiar ciegamente.

Papa, de tu experienca y conocimiento, tanto en la vida como en la investigación, he sacado fuerzas e ideas para seguir adelante durante todo el proyecto.

Rodrigo and Charlotte, because without movies and soup, this would have been tougher.

A Appendices

A.1 Tables Schemas

Trips

Column Name	Data Type	Description
VendorId	Integer	Code indicating the TPEP provider that provided the record.
pickup_datetime	Date	Date and time when the meter was engaged.
pdropoff_dateime	Date	Date and time when the meter was disengaged.
passenger_count	Integer	Number of passengers in the vehicle.
trip_distance	Float	The elapsed trip distance in miles.
pickup_longitude	Float	Longitude where the meter was engaged.
pickup_latitude	Float	Latitude where the meter was engaged.
Rate_CodeId	Integer	The final rate code in effect at the end of the trip
Flag	String	Flag indicates whether the trip record was held in vehicle memory before sending to the vendor.
dropoff_longitude	Float	Longitude where the meter was disengaged
dropoff_latitude	Float	Latitude where the meter was disengaged
payment_type	Integer	Numeric code signifying how the passenger paid for the trip.
fare_amount	Float	The time-and-distance fare calculated by the meter.
extra	Float	Extras and surcharges.
mta_tax	Float	$0.50 MTA tax that is automatically charged.
tolls_amount	Float	Total amount of all tolls paid in trip.
improvement	Float	If there was an improvement surcharge and how much.
total_amount	Float	The total amount charged to the passengers.

Table 32: Schema for Trips dataset

Trips_split

Column Name	Data Type	Description
VendorId	Integer	Code indicating the TPEP provider that provided the record.
pickup_datetime	Date	Date and time when the meter was engaged.
pdropoff_dateime	Date	Date and time when the meter was disengaged.
passenger_count	Integer	Number of passengers in the vehicle.
trip_distance	Float	The elapsed trip distance in miles.
pickup_longitude	Float	Longitude where the meter was engaged.
pickup_latitude	Float	Latitude where the meter was engaged.
Rate_CodeId	Integer	The final rate code in effect at the end of the trip
Flag	String	Flag indicates whether the trip record was held in vehicle memory before sending to the vendor.
dropoff_longitude	Float	Longitude where the meter was disengaged
dropoff_latitude	Float	Latitude where the meter was disengaged
payment_type	Integer	Numeric code signifying how the passenger paid for the trip.
fare_amount	Float	The time-and-distance fare calculated by the meter.
extra	Float	Extras and surcharges.
mta_tax	Float	$0.50 MTA tax that is automatically charged.
tolls_amount	Float	Total amount of all tolls paid in trip.
improvement	Float	If there was an improvement surcharge and how much.
total_amount	Float	The total amount charged to the passengers.

Table 33: Schema for Trips_split dataset

Neighborhoods

Column Name	Data Type	Description
BoroCode	Integer	Code for every Borough.
BoroName	String	Borough's name.
NTACode	Integer	National Transport Association Code.
NTAName	String	National Transport Association Name.
geom	Geometry	The polygons

Table 34: Schema for Neighborhoods dataset

Zones

Column Name	Data Type	Description
ZONEDIST	String	Zone's code.
geom	Geometry	The polygons.

Table 35: Schema for Zones dataset

Streets

Column Name	Data Type	Description
Jurisdiction	Integer	Code for the state to which the segment belongs.
LeftCounty	String	County located to the west of the segment.
RightCounty	String	County located to the east of the segment.
LeftCityTo	String	City located to the west of the segment.
RightCityTo	String	County located to the east of the segment.
Label	String	What type of entity the segment is.
geom	Geometry	The different lines.

Table 36: Schema for Streets dataset

Gewaspercelen

Column Name	Data Type	Description
Year	Integer	The year that geometry was entered.
Grondgebruik	String	The land use.
crop_code	Integer	A code that defines the crop.
crop_name	String	The crop name
geom	Geometry	The Polygons.

Table 37: Schema for Gewaspercelen dataset

Bofek

Column Name	Data Type	Description
ObjectID	Integer	The global id.
bofek2012	Integer	The code for the lookup table.
geom	Geometry	The geometries.

Table 38: Schema for Bofek dataset

References

Amazon Web Services. Amazon ec2. https://aws.amazon.com/ec2, 2017a.

Amazon Web Services. Amazon ec2 instance types. https://aws.amazon.com/ec2/instance-types/, 2017b.

Amazon Web Services. Amazon educate starter account. https://www.awseducate.com/microsite/, 2017c.

Apache Accumulo. Apache accumulo. https://accumulo.apache.org/, 2017.

Apache Casandra. Apache casandra. https://cassandra.apache.org/, 2017.

Apache HBase. Apache hbase. https://hbase.apache.org/, 2017.

Apache Kafka. Apache kafka. https://kafka.apache.org/, 2017.

M Armbrust, Y Huai, C Liang, R Xin, and M Zaharia. Deep dive into spark sql catalyst optimizer. https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html, 2015.

Autodesk. Autodesk. https://www.autodesk.com/, 2015.

Azavea. Azavea. https://www.azavea.com/, 2017.

Benjamin Bengfort and Jenny Kim. *Data Analytics with Hadoop: An Introduction for Data Scientists*. " O'Reilly Media, Inc.", 2016.

Bentley. Microstation. https://www.bentley.com/en/products/brands/microstation, 2017.

CCRI. Ccri. http://www.ccri.com/.

Nicholas Chammas. Flintrock, 2018.

K.-T Chang. *Introduction to geographic information systems*. "McGraw-Hill Higher Education.", 2004.

Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

Shane Cook. CUDA, C Programming guide. *NVIDIA, July*, 2013.

Databricks. Databricks. https://databricks.com/, 2017a.

Databricks. Spark csv. https://github.com/databricks/spark-csv, 2017b.

ESRI. Esri. https://www.esri.com/en-us/home, 2016.

Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

GDAL. Gdal. http://www.gdal.org/, 2016.

Geo Tools. Geo tools. http://www.geotools.org/, 2016.

GeoPySpark. Introducing geopyspark, 2017.

GeoPySpark. Geopyspark docs v0.3.0. https://media.readthedocs.org/pdf/geopyspark/latest/geopyspark.pdf, 2018.

Geospark. Geospark. "http://geospark.datasyslab.org/", 2017.

GeoTrellis. Geopyspark. https://github.com/locationtech-labs/geopyspark", 2017.

GIS Dictionary ESRI. Gis dictionary. http://support.esri.com/en/other-resources/gis-dictionary/, 2015.

GIS Geography. Gis geography blog. http://gisgeography.com/, 2017.

GIS Lounge. Open source big spatial data with geomesa, 2015.

GRASS Development Team. *Geographic Resources Analysis Support System (GRASS GIS) Software*. Open Source Geospatial Foundation, USA, 2015.

Daniel A Griffith, Yongwan Chun, and Denis J Dean. Advances in geocomputation. 2017.

Qingfeng Guan and Keith C Clarke. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science*, 24(5):695–722, 2010.

Qingfeng Guan, Phaedon C Kyriakidis, and Michael F Goodchild. A parallel computing approach to fast geostatistical areal interpolation. *International Journal of Geographical Information Science*, 25(8):1241–1267, 2011.

Christophe Gurret, Yannis Manolopoulos, Apostolos N Papadopoulos, and Philippe Rigaux. The basis system: a benchmarking approach for spatial index structures. In *Spatio-Temporal Database Management*, pages 152–170. Springer, 1999.

gvSIG. Generalitat valenciana sig. `http://www.gvsig.com/en/home`, 2015.

Hadoop. Apache hadoop. `"http://hadoop.apache.org/"`, 2017.

Harris Geospatial. Envi. `http://www.harrisgeospatial.com/ProductsandTechnology/Software/ENVI.aspx`, 2016.

Atallah Hezbor and Jim Hughes. Maritime location intelligence with exactearth data and geomesa, 2017.

Zhou Huang, Yiran Chen, Lin Wan, and Xia Peng. Geospark sql: An effective framework enabling spatial queries on spark. 6:285, 09 2017a.

Zhou Huang, Yiran Chen, Lin Wan, and Xia Peng. Geospark sql: An effective framework enabling spatial queries on spark. *ISPRS International Journal of Geo-Information*, 6(9):285, 2017b.

Capers Jones. *Applied software measurement: global analysis of productivity and quality*. McGraw-Hill Education Group, 2008.

Rob Knapen.

Jae-Gil Lee and Minseo Kang. Geospatial big data: challenges and opportunities. *Big Data Research*, 2(2): 74–81, 2015.

Kisung Lee, Raghu K. Ganti, Mudhakar Srivatsa, and Ling Liu. Efficient spatial query processing for big data. In *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '14, pages 469–472, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3131-9. doi: 10.1145/2666310.2666481. URL `http://doi.acm.org/10.1145/2666310.2666481`.

Rakesh K Lenka, Rabindra K Barik, Noopur Gupta, Syed Mohd Ali, Amiya Rath, and Harishchandra Dubey. Comparative analysis of spatialhadoop and geospark for geospatial big data analytics. In *Contemporary Computing and Informatics (IC3I), 2016 2nd International Conference on*, pages 484–488. IEEE, 2016.

G Leptoukh. Nasa remote sensing data in earth sciences: Processing, archiving, distribution, applications at the ges disc. In *Proc. of the 31st Intl Symposium of Remote Sensing of Environment*, 2005.

Songnian Li, Suzana Dragicevic, Francesc Antón Castro, Monika Sester, Stephan Winter, Arzu Coltekin, Christopher Pettit, Bin Jiang, James Haworth, Alfred Stein, et al. Geospatial big data handling theory and methods: A review and research challenges. *ISPRS Journal of Photogrammetry and Remote Sensing*, 115:119–133, 2016.

Arend Ligtenberg. Data handling, alterations. 2016.

Loation Tech. Location tech. `https://www.locationtech.org/`, 2017.

LocationTech. Java topology suite.

Ping Lu, Nicola Casagli, Filippo Catani, and Veronica Tofani. Detecting hot spots using cluster analysis & gis. *International Journal of Remote Sensing*, 33(2):466–489, 2012.

Magellan. Magellan. "https://github.com/harsha2010/magellan", 2017.

John C Mankins. Technology readiness levels. *White Paper, April*, 6, 1995.

Marble. Marble. https://marble.kde.org/, 2014.

Apache Maven. Apache maven packages. "https://mvnrepository.com/", 2017.

Martin Molenaar. *An introduction to the theory of spatial object modelling for GIS*. CRC Press, 1998.

Jussi Myllymaki and James Kaufman. Dynamark: A benchmark for dynamic spatial indexing. In *Mobile data management*, pages 92–105. Springer, 2003.

Open Geospatial Consortium. Opengis® implementation specification for geographic information-simple feature access-part 1: Common architecture. *OGC document*, 2011.

Oracle. Oracle docs, 2017.

OSGeo. Proj.4 docs. http://proj4.org/, 2016.

Oxford English Dictionary. Oxford english dictionary online, 2007.

Rabi Prasad Padhy. Big data processing with hadoop-mapreduce in cloud systems. *International Journal of Cloud Computing and Services Science*, 2(1):16, 2013.

Norman Paton, M Williams, Kosmas Dietrich, Olive Liew, Andrew Dinn, and Alan Patrick. Vespa: A benchmark for vector spatial databases. *Advances in Databases*, pages 81–101, 2000.

Alan Phillips, Frank Schroth, Geoffrey M Palmer, Stefan G Zielinski, Allen P Smith, and Colin M Cunningham III. Location-based services, December 7 2010. US Patent 7,848,765.

PostgreSQL. Postgis. http://postgis.net/, 2015.

QGIS. Quantum gis. http://www.qgis.org/en/site/, 2016.

Suprio Ray, Bogdan Simion, and Angela Demke Brown. Jackpine: A benchmark to evaluate spatial database performance. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1139–1150. IEEE, 2011.

Arthur H Robinson. Elements of cartography. *Soil Science*, 90(2):147, 1960.

G. L. Sanders and Seungkyoon Shin. Denormalization effects on performance of rdbms. pages 9–15, 2001.

Pedro O. Santos, Mirella M. Moro, and Clodoveu A. Davis. Comparative performance evaluation of relational and nosql databases for spatial and mobile applications. pages 186–200, 2015.

Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

James G Shanahan and Laing Dai. Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2323–2324. ACM, 2015.

Shapely. Shapely docs. https://pypi.python.org/pypi/Shapely, 2017.

Shashi Shekhar, Viswanath Gunturi, Michael R Evans, and KwangSoo Yang. Spatial big-data challenges intersecting mobility and cloud computing. In *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 1–6. ACM, 2012.

Spark. Apache Spark. https://spark.apache.org/, 2017a.

Apache Spark. Spark packages. "https://spark-packages.org/", 2017b.

Ram Sriharsha. How does magellan scale geospatial queries. https://magellan.ghost.io/how-does-magellan-scale-geospatial-queries/, 2017.

Michael Stonebraker, Jim Frew, Kenn Gardels, and Jeff Meredith. The sequoia 2000 storage benchmark. In *ACM SIGMOD Record*, volume 22, pages 2–11. ACM, 1993.

Ian Turton and Stan Openshaw. High-performance computing and geography: Developments, issues, and case studies. *Environment and Planning A*, 30(10):1839–1856, 1998.

S Vasavi, M Padma Priya, and Anu A Gokhale. Framework for geospatial query processing by integrating cassandra with hadoop. In *Knowledge Computing and Its Applications*, pages 131–160. Springer, 2018.

Kewen Wang and Mohammad Maifi Hasan Khan. Performance prediction for apache spark platform. pages 166–173, 2015.

Jia Yu, Jinxuan Wu, and Mohamed Sarwat. A demonstration of geospark: A cluster computing framework for processing big spatial data. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1410–1413. IEEE, 2016.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

Daniel Zeng, Hsinchun Chen, Robert Lusch, and Shu-Hsing Li. Social media analytics and intelligence. *IEEE Intelligent Systems*, 25(6):13–16, 2010.