Benchmarking Apache Spark spatial libraries

Hector Muro Mauri
Wageningen University and Research, hector.muro@wur.nl

Rob Knapen
Wageningen University and Research, rob.knapen@wur.nl

Arend Ligtenberg
Wageningen University and Research, Arend.Ligtenberg@wur.nl

Sander Janssen
Wageningen University and Research, sander.janssen@wur.nl

Ioannis N. Athanasiadis
Wageningen University and Research, ioannis@athanasiadis.info

iEMSs International Congress on Environmental Modelling and Software

9th International Congress on Environmental Modelling and Software
Fort Collins, Colorado, USA, Mazdak Arabi, Olaf David, Jack Carlson, Daniel P. Ames (Eds.)
https://scholarsarchive.byu.edu/iemssconference/2018/

Benchmarking Apache Spark spatial libraries

Hector Muro[a], Arend Ligtenberg[a], Rob Knapen[a], Sander Janssen[a], Ioannis N. Athanasiadis[a]

[a] *Wageningen University & Research, Wageningen, The Netherlands*
hector.muro@wur.nl;arend.ligtenberg@wur.nl;rob.knapen@wur.nl;sander.janssen@wur.nl
ioannis@athanasiadis.info

Abstract: Apache Spark is one of the most widely used and fast-evolving cluster-computing frameworks for big data. This research investigates the state of practice in the Apache Spark ecosystem for managing spatial data, with a specific focus on spatial vector data. Apache Spark is a relatively new platform, and the associated libraries for geospatial data extensions are still work-in-progress. In this work, three libraries for managing geospatial information in Apache Spark have been investigated, namely GeoSpark, GeoPySpark, and Magellan. First we designed and performed a suite of functionality tests, to explore how much can be done with. Then, we benchmarked the performance of the libraries for executing common spatial tasks using *annoyingly* big geospatial datasets. Finally, we compare the performance of the three libraries in contrast to a traditional Geographic Information System that uses a relational database for storage. Our findings about the maturity of the libraries and the scalability of solutions in Apache Spark are mixed, as key functionalities are still missing, but gains in the elapsed real time to respond to queries can be up to two orders of magnitude faster.

***Keywords**:* Apache Spark; GIS; Big Geospatial Vector Data; Evironmental modelling.

1 INTRODUCTION

Over the last decade, the amount of data generated every day has not stopped increasing as well as the source types where data originate from. Estimations report that 2.5 quintillion bytes of data are generated every day and a large portion of it is related to a geographical location [Lee and Kang, 2015]. For example, the NASA Earth Observing System generates one terabyte of data every day [Leptoukh, 2005]. The location-related big data is commonly refered to as Geospatial Big Data [Lee and Kang, 2015; Li et al., 2016]. Geospatial Big Data can be defined as 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].

The typical tool for processing geospatial data is a Geographic Information System (GIS), which is a comprehensive technology involving geography, mapping science, computer science, and other disciplines. With the progress in computer science and the evolution in computer based environmental modelling, GIS systems change constantly. Going from desktop GIS (1960s) to the Web GIS (1980s), and the distributed GIS (1990s), to the Cloud GIS (2010s). Next, it will need to enter this era of Big Data. Which already has a certain influence on GIS, e.g. more focus in recent versions of GIS software for efficiently making use of multi-core processors, but has of yet not changed the basic pradigms behind GIS [Yao and Li, 2018].

One of the core components of current GIS is a database that handles the storage and retrieval of the geospatial data. Usually this is some variant of a Relational Database Management System (RDBMS). However, traditional RDBMS can not cope anymore with the volume and variety of Big Data, nor the speed at which it is generated. This is mostly because of it's own definition constraints, such as ACID (Atomicity, Consistency, Isolation, Durability) and SQL features [Moniruzzaman, 2014]. Thus, the Big Data era calls for a shift in the way the data is managed. Examples are parallel distributed systems, and various NoSQL (Not only SQL) databases.

In the case of Geospatial Big Data, there are several reported works trying to extend Big Data tools, as Hadoop Distributed File System (HDFS) and MapReduce (a programming model for processing and generating big data sets with a parallel, distributed algorithm on a cluster), for supporting geospatial infromation. For example, Chen et al. [2013] employed Hadoop to implement a cloud-based spatial storage system. ESRI has also developed a Toolkit for Hadoop containing a Geometry API (Application Programming Interface) [ESRI, 2017]. By using Hadoop MapReduce, processing capabilities can be improved by parallelizing spatial queries on top of the HDFS using MapReduce (i.e. dividing the query in parts (the 'map' phase), run the partial queries on several computers at the same time, and combine the results (the 'Reduce' phase), as Spatial Hadoop [Eldawy and Mokbel, 2015], and Hadoop GIS [Aji et al., 2013]. However, because Hadoop MapReduce produces a lot of I/O operations, the performance achieved in geospatial applications is not promising [Huang et al., 2017a].

On the other hand, Apache Spark is a cluster-computing framework for distributed programming, that can operate on top of HDFS and MapReduce, using a unified API. Apache Spark performs significantly faster than Hadoop MapReduce due to its in-memory processing and the support for more complex operator graphs [Zaharia et al., 2016]). Possibly these are some of the reasons on its increasing popularity among big data developers. Also, Apache Spark ensures *scalability* and *fault tolerance*. In Apache Spark any algorithm can run no matter how big the data set is (vertical scalability) or how many processors are used (horizontal scalability). Fault tolerance stands for avoiding data loss in case a cluster node in the system would fail.

There are a few projects that use Apache Spark for big geospatial data processing. For example, Spatial Spark [You et al., 2015] demonstrated two different spatial joins, broadcast spatial join and partitioned spatial join, based on the dataset sizes. Magellan [Sriharsha, 2017b], currently under development, works towards extending the Apache Spark dataframe abstraction to conduct spatial analysis [Sriharsha, 2017a]. GeoSpark extends the Resilient Distributed Datasets (RDD) abstraction to Spatial RDD to enable spatial querying in Apache Spark. The libraries above focus only on vector data. There are also libraries working with raster data, most notably GeoTrellis, that provides a number of operations to manipulate raster data, including cropping/warping, as well as map algebra operations.

All Apache Spark geospatial libraries are still quite new and under development. In the contrary, GIS tools have been developed for decades, which translates to efficient algorithms and performance [Dunn and Newton, 1992; Leutenegger et al., 1997]. Our objective with this work is to assess whether Apache Spark spatial libraries are able to provide scalable geospatial data processing over a cluster. Therefore, we benchmarked geospatial Apache Spark geospatial libraries and compare them with the functionality and performance of a traditional GIS system, with a particular focus on vector data. This will allow us to investiage how close are they in becoming useful for a geo-information scientist.

Specifically, two different test suits have been defined. First, a *functionality benchmark test* was designed to verify which (spatial) operations are supported. We reviewed the literature for existing benchmarks for geoinformation systems, and combined our findings with a review of the basic GIS operations, in order to define a minimum set of spatial operations that should be needed by a geo-information scientist.

Next to that, a *performance benchmark test* was designed to measure the efficiency of geospatial processing with Apache Spark. Inspired from previous work that included testing the performance of geospatial queries [Huang et al., 2017b], we defined and applied a set of tests to measure the performance of Apache Spark geospatial libraries and how this compares with ”traditional” software.

We conducted our tests in three libraries: Magellan, GeoSpark and GeoPySpark, and compared their performance against PostgreSQL.

2 FUNCTIONALITY BENCHMARK TEST

2.1 Background & Definition

In order to define a functionality test, we first reviewed the literature for geospatial data benchmarking frameworks. One of the first Geospatial Data Benchmark tests defined was SEQUOIA 2000 [Stonebraker et al., 1993], which is inspired from Earth Sciences and focuses on raster data. Following SEQUOIA 2000, other spatial data benchmark tests research have been reported, as Jackpine [Ray et al., 2011] that evaluates spatial database performance. Jackpine is divided in two smaller benchmarks: one to test topological relationships and another to test geospatial services. VESPA [Paton et al., 2000] is a benchmark for vector databases, designed to assess a wide range of functionality.

The previous tests provide with guidance on the general structure that a spatial benchmark test should follow. Following the suggestion of [Paton et al., 2000] that a benchmark test should contain a wide range of functionality, we revisited Chang [2004], that classifies the basic GIS analytic operations in the following three blocks: a. Data exploration, b. Vector Data Analytics, c. 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 analytics refers to the data that uses coordinates to create spatial features such as points, lines and polygons. One of the key aspects is topology [Molenaar, 1998], i.e. spatial relationships between features. This is very important because it defines the relative location of an object and ultimately any kind of operation on geospatial data. Topology serves also to create bet-

ter 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 operations**: Measuring straight (Euclidean) lines between features, geodetic distances or spheric distances.

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; i.e. 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 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[1]:
1. **Local**: These operations happen at a cell level, the value in the output raster is at the same location on the input raster.
2. **Focal**: 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**: 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).
4. **Global**: This type of operations are those that involve the whole raster. Euclidean distances or cost distances are examples of global operations in raster.

Based on the spatial benchmark tests reviewed and the basic spatial functionality frameworks of Chang [2004] and Molenaar [1998], we defined a set of twenty functionality tests for big geospatial data which is summarized in Table 1. A detailed definition of the functionality tests is documented in Muro [2018].

2.2 Protocol & Datasets

In order to conduct the twenty functionality tests, we extensively studied the documentation of the three libraries, and/or reviewed their source code. All the tests have been implemented in Apache Spark independently using the Amazon Web Services or the Databricks environments. In order to facilitate easy validation of the results we created a set of reference data with simple geometries/grids, with which basic functionality tests were implemented. The exact same tests and data were applied in all libraries. In the case of vector data, points, lines and polygons were created using the code constructors of the libraries. Similarly for the raster data, a squared matrix 4 by 4 filled with 1's was defined as a simple reference dataset.

[1]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]

Operation		Description	Magellan	GeoPySpark	GeoSpark
Reading Data		Accepted file formats	0.35	0.5	0.5
Queries	Selection by attributes	Thematic Data Query	1	1	1
		Raster Data Query		1	
	Spatial	Geometric Data Query	1	1	1
	Temporal	Temporal Data Query		1	
Transformations	Coordinate System	Transform from a coordinate system to another	1	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			1
	Intersects	Feature intersects feature	0.8		1
	Within	Feature within feature	0.5		1
	Contains	Feature contains Feature	0.6		1
	Union	Union of two features that share spatial extent			1
(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			4.5/20	9.5/20	8.5/20

Table 1: Functionality Benchmark Test Definition & Results

2.3 Results

Table 1 shows next to the definition of the functionality benchmark test, the results obtained from applying it to the different libraries. When the operation was defined and fully functional, a whole point was given. There were some cases were some implementations were missing, e.g. Magellan has defined three predicates; *Intersects, Within and Contains*, but these operations are not valid for all data pairs (Point-Point, Point-Line, Line-Polygon, etc.) and this is why not the whole point was given.

3 PERFORMANCE BENCHMARK TEST

3.1 Background & Definition

There are many ways of comparing software performances. [Huang et al., 2017b] defined a set of queries that were performed in PostGIS, an SQL extension of GeoSpark defined by the authors of the paper, and ESRI spatial framework for Hadoop. For every type of queries different queries were defined and every query was ran 10 times, taking the average time as the final time. In this paper, we followed the structure and results from the functionality benchmark test, the operations to be tested have been divided among three groups: Queries, Transformations and Alterations.

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. This set of queries have been subdivided into two groups. The first one query text files and the second one spatial data files (shapefiles).
Containment Query: Two queries to return all the points that fall within a dataset of polygons.
Intersection Query: Two intersection queries between different geometries. (Line - Polygon).
KNN Query: Two queries that returns the closest k features, 1000 in this particular case, from a given feature, e.g. a point.

Transformations
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).

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 an alteration, which modifies the thematic and/or geometric definition of the original data set.

In our experimental setup, we deployed an Apache Spark cluster of 4 nodes on Amazon Web Services, each with 2.3 GHz Intel Xeon E5-2686 v4 processor (18 cores), 6 GB RAM and 30 GB hard disk. Apache Spark 2.0 was deployed with Magellan version 1.0.6, GeoSpark version 1.0.1. PostgreSQL version 9.1 was deployed on a personal computer with Linux Ubuntu 16.04.4, Intel Core i7 processor (4 cores) and 8 Gb RAM and 500 Gb hard disk. Note that PostgreSQL only started adding limit parallel processing in version 9.6.

3.2 Protocol & Datasets

Every query has been run 10 times and the average time has been taken as the reported time. Other times like the loading time (for every dataset) or the warm-up query time (the first time an operation was run was not taken into account in the final average, but measured separately), have also been reported.

Data used for the Performance Benchmark Test was selected to be open data, summarized in Table 2. As the funcntionality test has demonstrated that only one library out of three was supporting raster operations, this research has not conducted the raster performance test.

In our experimental setup, we deployed an Apache Spark cluster of 4 nodes on Amazon Web Services, each with 2.3 GHz Intel Xeon E5-2686 v4 processor (18 cores), 6 GB RAM and 30 GB hard disk. Apache Spark 2.0 was deployed with Magellan version 1.0.6, GeoSpark version 1.0.1. PostgreSQL version 9.1 was deployed on a personal computer with Linux Ubuntu 16.04.4, Intel Core i7 processor (4 cores) and 8 Gb RAM and 500 Gb hard disk. Note that PostgreSQL only started adding limit parallel processing in version 9.6.

Dataset Name	Data Type	Description	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.[a]
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[b]
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 2: Performance Test Data sets

[a]http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
[b]http://www1.nyc.gov/site/planning/data-maps/open-data.page

3.3 Results

A detailed report of the results obtained is documented in Muro [2018]. Figure 1 illustrates the results of two attribute queries against two point datasets, originally stored in CSV. The tests were executed on the datasets Trips_split and Trips. Since the one dataset is a slice of the other, we evaluate the scalability performance, as the number of points in the dataset increases from 65 thousands to 12 million. In Figure 1, column 1 and 3 are the results of the querying for the first attribute, but 1 is queried to Trips_split and 3 to Trips. Querying for the second attribure is presented in columns 2 and 4.

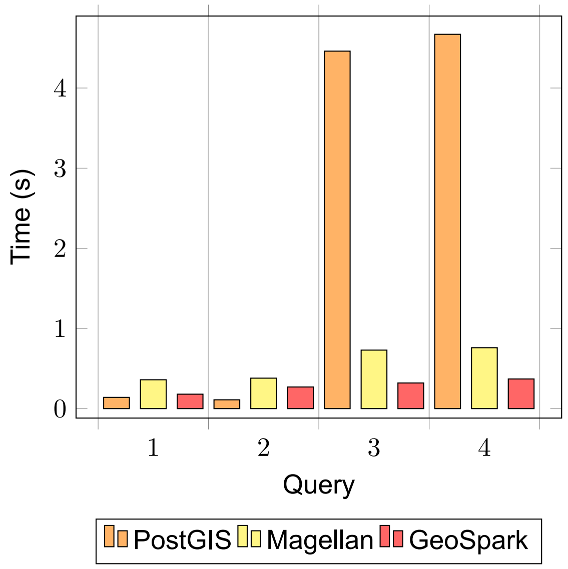

Figure 1: Attribute queries wall time (sec) - average 10 repetitions

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 3: Attribute queries wall time (sec). Reporting average accross 10 repetitions - st.dev. in brackets

Figure 2 presents the wall times recorded when executing two intersection queries. Both are Line-Polygon queries, what changes is the size of the polygon dataset. Query 1 is between Streets dataset and Neighborhoods dataset with 192 polygons, whilst Query 2 is between Streets dataset and Zones dataset with 4500 polygons.

Finally, Figure 3 illustrates the average wall time to transform each of the datasets used for this test into another coordinate system. The datasets are ordered by incremental size. This, together with the logarithmic y-axes provides a picture on how does performance evolve with bigger or more complex datasets.

4 DISCUSSION

From the results obtained during the functionality test, we conclude that the existing functionality of the Apache Spark spatial libraries is not yet developed enough to cope with the whole spectrum of a geo-information scientist needs. Not solely the lack of operations, but also the lack of some topological relations, illustrates that key functionalities are still missing. At the same time, the final grade in our functionality test should be read carefully. All three Apache Spark spatial libraries we evaluated are meant to be Raster-only or Vector-only. Thus, the maximum grade they could obtain is 10 out of 20. This is one of the reasons of the low grades all libraries obtained in the functionality test.

The attribute queries performance test results prove Apache Spark power in scaling. Although incrementing file sizes more than 180 times, the query wall time approximately doubles in Apache Spark with Magellan.In the contrary, it requires 40-times more time with PostgreSQL.

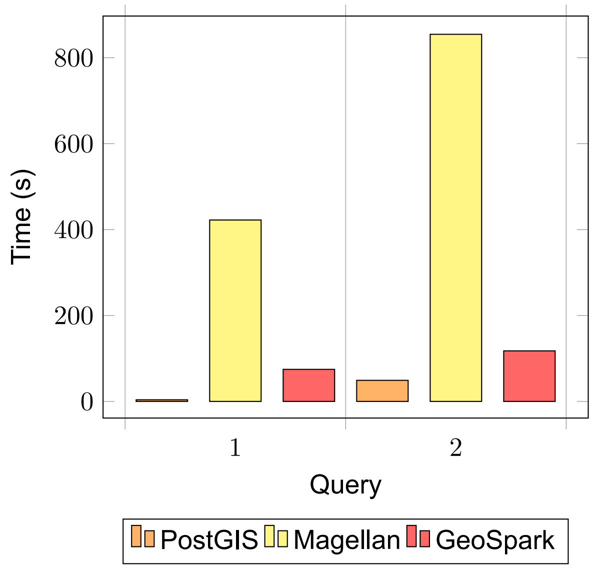

Figure 2: Intersection Query Times (s), 10 repetitions

No.	**PostGIS**	**Magellan**	**GeoSpark**
1	3.835 (0.85)	422.32 (5.45)	54.16 (1.52)
2	49.1 (1.29)	854.58 (8.34)	117.67 (1.80)

Table 4: Intersection Query Times (s), 10 repetitions

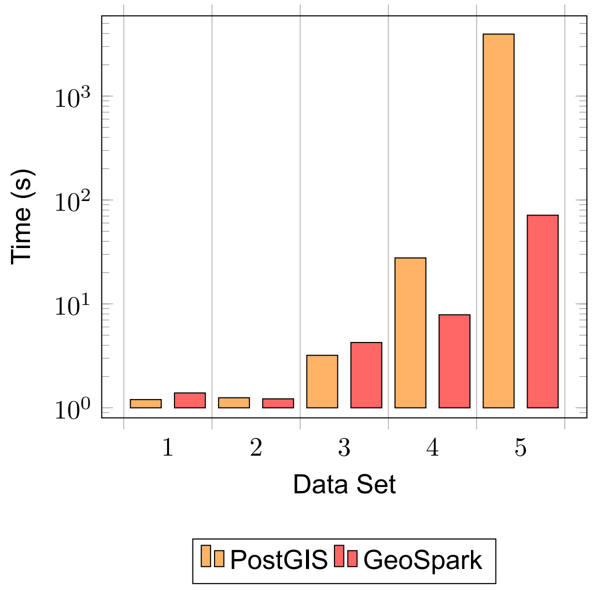

Figure 3: Transformation Times (s), 10 repetitions

Size	Dataset	**PostGIS**	**GeoSpark**
3.4 Mb	Neighborhoods	0.70 (0.01)	0.86 (0.02)
6.1 Mb	Zones	0.71 (0.02)	0.72 (0.03)
10 Mb	Trips_split	3.2 (0.02)	4.25 (0.89)
150 Mb	Streets	27.7 (0.09)	7.86 (0.05)
1.8 Gb	Trips	3950 (2.31)	71.32 (1.25)

Table 5: Transformation Times (s), 10 repetitions

In the intersection query times, Magellan's performance was very dissapointing. 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. Also, we noted that there was no intersection query where Apache Spark libraries outperformed PostGIS. 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 optimized than the algorithms implementations in Magellan and GeoSpark.

Last, the transformation query results indicate that the more complex an operation is the more benefits are to be expected from parallelizing. The logarithmic y-axes show how the wall time scales better in Apache Spark compared to PostGIS.

With the work reported here, and from our experience during performing it we conclude that Apache Spark spatial libraries are still lagging behind when compared to "traditional" GIS software. At the same time, the performance results reported indicate that the bigger or more complex a spatial operation is, the bigger become the performance gains of using an Apache Spark library. This underlines the potential of Apache Spark spatial libraries for more efficient big geospatial data processing. And, when having to handle truly Big Data, such software packages are the only option available at the moment.

BIBLIOGRAPHY

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

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

Chen, C., J. Lin, X. Wu, J. WU, and H. LIAN (2013). Massive geospatial data cloud storage and services based on nosql database technique. *Journal of Geo-Information Science 15*(2), 166–174.

Dunn, C. E. and D. Newton (1992). Optimal routes in gis and emergency planning applications. *Area 24*(3), 259–267.

Eldawy, A. and M. F. Mokbel (2015). SpatialHadoop: A MapReduce Framework for Spatial Data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pp. 1352–1363.

ESRI (2017). Gis tools for hadoop. `https://esri.github.io/gis-tools-for-hadoop/`.

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

Huang, Z., Y. Chen, L. Wan, and X. Peng (2017a). Geospark sql: An effective framework enabling spatial queries on spark. *ISPRS International Journal of Geo-Information 6*(9), 285.

Huang, Z., Y. Chen, L. Wan, and X. Peng (2017b, September). Geospark sql: An effective framework enabling spatial queries on spark. *ISPRS International Journal of Geo-Information 6*, 285.

Lee, J.-G. and M. Kang (2015). Geospatial big data: challenges and opportunities. *Big Data Research 2*(2), 74–81.

Leptoukh, G. (2005). 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*.

Leutenegger, S. T., M. A. Lopez, and J. Edgington (1997). Str: A simple and efficient algorithm for r-tree packing. In *Data Engineering, 1997. Proceedings. 13th international conference on*, pp. 497–506. IEEE.

Li, S., S. Dragicevic, F. A. Castro, M. Sester, S. Winter, A. Coltekin, C. Pettit, B. Jiang, J. Haworth, A. Stein, et al. (2016). Geospatial big data handling theory and methods: A review and research challenges. *ISPRS Journal of Photogrammetry and Remote Sensing 115*, 119–133.

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

Moniruzzaman, A. (2014). Newsql: Towards next-generation scalable rdbms for online transaction processing (oltp) for big data management. *arXiv preprint arXiv:1411.7343*.

Muro, H. (2018). Managing big geospatial data with Apache Spark. Msc thesis, Wageningen University.

Paton, N., M. Williams, K. Dietrich, O. Liew, A. Dinn, and A. Patrick (2000). Vespa: A benchmark for vector spatial databases. In *Advances in Databases*, pp. 81–101. Springer.

Ray, S., B. Simion, and A. D. Brown (2011). Jackpine: A benchmark to evaluate spatial database performance. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 1139–1150. IEEE.

Shekhar, S., V. Gunturi, M. R. Evans, and K. Yang (2012). 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*, pp. 1–6. ACM.

Sriharsha, R. (2017a). How does magellan scale geospatial queries. `https://magellan.ghost.io/how-does-magellan-scale-geospatial-queries/`.

Sriharsha, R. (2017b). Magellan. `https://github.com/harsha2010/magellan`.

Stonebraker, M., J. Frew, K. Gardels, and J. Meredith (1993). The sequoia 2000 storage benchmark. *ACM SIGMOD Record 22*(2), 2–11.

Yao, X. and G. Li (2018). Big spatial vector data management: a review. *Big Earth Data 2*, 108–129.

You, S., J. Zhang, and L. Gruenwald (2015). Large-scale spatial join query processing in cloud. In *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pp. 34–41. IEEE.

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