NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES
”ADVANCED INFORMATION SYSTEMS”

MSc THESIS

Experimental Evaluation of Big Geospatial Data Systems

Konstantinos E. Giannousis

SUPERVISORS: **Manolis Koubarakis**, Professor
Konstantina Bereta, PhD Candidate

ATHENS

SEPTEMBER 2018

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
”ΠΡΟΗΓΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πειραματική Αξιολόγηση Συστημάτων Γεωχωρικών Δεδομένων Μεγάλης Κλίμακας

Κωνσταντίνος Ε. Γιαννούσης

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανώλης Κουμπαράκης**, Καθηγητής
Κωνσταντίνα Μπερέτα, Υποψήφια Διδάκτωρ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2018

MSc THESIS

Experimental Evaluation of Big Geospatial Data Systems

Konstantinos E. Giannousis
Register Number: M1400

SUPERVISORS: **Manolis Koubarakis**, Professor
Konstantina Bereta, PhD Candidate

EXAMINATION COMMITTEE:

Manolis Koubarakis, Professor

Examination Date:

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πειραματική Αξιολόγηση Συστημάτων Γεωχωρικών Δεδομένων Μεγάλης Κλίμακας

Κωνσταντίνος Ε. Γιαννούσης
Α.Μ.: Μ1400

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανώλης Κουμπαράκης**, Καθηγητής
Κωνσταντίνα Μπερέτα, Υποψήφια Διδάκτωρ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Μανώλης Κουμπαράκης, Καθηγητής

Ημερομηνία Εξέτασης:

ABSTRACT

The volume of available spatial data that is generated and collected is significantly increased. A number of applications based on Map-Reduce-like systems and cloud infrastructure emerged. These applications offer a variety of features, however they differ in terms of spatial functions, partitioning and indexing. In this diploma thesis we present an experimental study that compares the most modern and complete systems of distributed geospatial query processing in terms of functionality and performance in runtime and scaling. We conduct detailed functional and performance benchmarks that include corner cases that stress the systems in comparison and reveal their advantages and weaknesses in both functionality and performance.

SUBJECT AREA: Geospatial Distributed Systems

KEYWORDS: Geospatial Data, Big Data, Spatial Data, Distributed Systems

ΠΕΡΙΛΗΨΗ

Ο όγκος των διαθέσιμων χωρικών δεδομένων ο οποίος παράγεται και συλλέγεται έχει αυξηθεί σημαντικά. Αναδύθηκε έτσι ένας αριθμός εφαρμογών που βασίζονται σε συστήματα τύπου Map-Reduce και υποδομών cloud. Αυτές οι εφαρμογές παρέχουν μια ποικιλία χαρακτηριστικών, παρόλα αυτά, διαφέρουν σε ότι αφορά τις χωρικές μεθόδους, τις τεχνικές καταμερισμού δεδομένων και το σύστημα ευρετηρίου που χρησιμοποιούν. Στην παρούσα διπλωματική εργασία παρουσιάζουμε μια πειραματική μελέτη που συγκρίνει τα πιο σύγχρονα και ολοκληρωμένα συστήματα κατανεμημένης επεξεργασίας γεωχωρικών επερωτήσεων ως προς την λειτουργικότητα και την απόδοση τους σε χρόνο εκτέλεσης και κλιμακωσιμότητα. Εκτελέστηκαν λεπτομερές συγκριτικές δοκιμές τόσο ως προς τις λειτουργίες όσο και στην απόδοση περιέχοντας οριακές περιπτώσεις που καταπονούν τα συγκρινόμενα συστήματα και αποκαλύπτουν τα πλεονεκτήματα και τις αδυναμίες τους τόσο λειτουργικά όσο και σε απόδοση.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κατανεμημένα Συστήματα Γεωχωρικών Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Γεωχωρικά Δεδομένα, Μεγάλα Δεδομένα, Χωρικά Δεδομένα, Κατανεμημένα Συστήματα

To my family and in memory of my beloved father.

ACKNOWLEDGEMENTS

I would like to thank my professor M. Koubarakis for giving me the opportunity to evolve in the field. I would like also to express a deep gratitude to PhD Candidate K. Bereta and my colleague N. Karalis for their help to implement this MSc thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

In this diploma thesis we present an experimental study that compares the most modern and complete systems of distributed geospatial query processing in terms of functionality and performance in runtime and scaling. We conduct detailed benchmarks that include corner cases to stress the systems in comparison and reveal their advantages and weaknesses in both functionality and performance.

1. INTRODUCTION

Large amounts of spatial data is currently available from multiple sources, GPS-enabled mobile phones, Sensor Observation Services, Automatic Identification System (AIS) sensors used in maritime and many more. This vast spatial data is collected and analyzed, as for example, the exploitation of a large volume of historical AIS data to estimate the location and connections of the major trade routes [12]. Meteorologists study and simulate climate data through spatial analysis. News reporters use geo-tagged tweets for event detection and analysis.

Domain experts (e.g., earth scientists, meteorologists, etc.) use extensively Geographic Information Systems (GIS) as they offer a user interface for visualising spatial data and performing spatial operations. Spatial data is often stored in geospatial databases. Decades of research focused in developing techniques of efficiently storing and querying geospatial data in relational databases. Most of these research results were adopted by RDBMSs and currently there is a variety of open source and commercial RDBMSs with geospatial support (e.g., PostGIS[1], Spatialite[2], Oracle-Spatial[3]).

In the era of big data and with the contributions of efforts made by multiple domains, the spatial data is first class citizen (e.g., Earth observation, AIS tracking etc). Developing techniques for distributed processing of spatial data is a major problem, that has been addressed by recent efforts that extend distributed frameworks (e.g., Spark[4], Hadoop[5]) with spatial support. The need for supporting spatial SQL queries makes this challenge even bigger.

This paper is organized as follows. In Chapter 2 we present the background knowledge about the geospatial systems. Then, in Chapter 3 we describe the current state-of-the-art in the area of distributed spatial data management systems which forms the ground on which our work is built on. Next, in Chapter 4, in section 4.1 we present the state-of-the-art distributed systems that offer spatial support (i.e., STARK[6], GeoSpark[7], Magellan[8], Spatial-Spark[9] and Exareme [3]) that take part to our evaluation, while in section 4.2 we perform a detailed functional and performance evaluation. Last, in section 4.3, we refer to the challenges that we run into during the benchmark. Finally, in Chapter 5 we summarise the findings of our study.

[1] https://postgis.net/
[2] https://www.gaia-gis.it/fossil/libspatialite/index
[3] https://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html
[4] https://spark.apache.org/
[5] http://hadoop.apache.org/
[6] https://github.com/dbis-ilm/stark
[7] http://datasystemslab.github.io/GeoSpark/
[8] https://github.com/harsha2010/magellan
[9] http://simin.me/projects/spatialspark/

2. BACKGROUND

Spatial objects, such as points, lines and polygons, in a Geographic Information System (GIS) are stored in a two-dimensional space and the queries asked of such a system are different from regular SQL queries. Typical geospatial queries [6] include partial matching queries, range queries, nearest-neighbor queries and where-am-I queries [11]. Thanks to the work of the Open Geospatial Consortium (OGC)[1], standards for spatial functions have been defined and are being adopted by many databases and, in our case, systems that are based in Map-Reduce approach and cloud infrastructure.

Additionally, since the spatial databases, the spatial indexes were introduced. Non-spatial RDBSs' indexes were not sufficient for spatial data. Spatial indexes allow systems to store the data in the file system in a spatial manner taking the spatial attributes into consideration [4]. The goal is to allow queries to run faster by making use of the index. The indexes implemented in systems vary and they include both flat indexes (grid and geohash) and hierarchical indexes (R-tree, R+-tree, Quad tree, PMR Quad tree and K-d tree). Notice that some systems implement local-only indexes by creating an index in each cluster node.

According to Gartner [2] *"Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation"*. This definition states that big data needs a new management system other than traditional databases, a system that is highly scalable, massively parallel and cost-effective. Hadoop has set the foundations for Map-Reduce-like systems and others extending it. In the Hadoop Ecosystem Table [2] are included a great number of such systems. It is noticeable that not many are mentioning spatial support.

In this paper we will not examine systems of the MapReduce architecture. Apache Spark is using RDD architecture and Exareme is a paralleDB system.

Apache Spark's architecture, that most systems in this paper support, is based on Resilient Distributed Datasets (RDDs) and Direct Acyclic Graphs (DAGs). RDDs are collections of data items that are split into partitions and can be stored in-memory on worker's nodes of the Spark cluster. They can support either Hadoop Datasets or parallelized collections based on existing Scala collections. DAGs are sequences of computations performed on data where each node is an RDD partition and edge is a transformation on top of data.

In a Parallel DB architecture a DBMS is extended in such way to provide a parallel DBMS system with a SQL-like query language.

In order to efficiently analyze and manage spatial data, state-of-the-art geospatial distributed systems emerged. Some of them are built on-top of existing systems for non-spatial data where user defined functions (UDFs) are used in order to handle spatial data [4]. Another approach is the build-in, where they use an existing system and they extend it by giving it spatial data awareness. There is of course a third approach, called from-

[1] http://www.opengeospatial.org/ogc
[2] https://hadoopecosystemtable.github.io

scratch where a new system is built. This is more efficient but also harder to maintain.

In the following lines we will examine systems that are extending RDDs into Spatial RDDs [1], collections that can manage better spatial datasets, or create their own spatial objects to support spatial data and operations [8]. Moreover, Exareme is extending spatial DBMS into a parallel spatial DMBS [3].

3. RELATED WORK

The first renowned systems for distributed spatial query processing were implemented as extensions for Hadoop MapReduce such as SpatialHadoop[1], Hadoop-GIS[2], and Hive on Hadoop[3] with ESRI's spatial extensions. Hadoop provides a very fault tolerant environment for parallel execution, but storing intermediate results to disk increases the execution time for spatial operations. Hence, the in-memory execution model of Spark became very popular as it reduces the execution time drastically, compared to MapReduce jobs [7]. Following to this trend, many Spark-based systems included geospatial support, most notable of which are the systems STARK, GeoSpark, Magellan and Spatial-Spark. In the context of this work, we will only consider the Spark-based systems as they reportedly achieve better performance [7, 8] than the Hadoop-based systems.

STARK [8] is built on top of Spark and provides a domain specific language (DSL) that seamlessly integrates into any Spark program (written in Scala). It also includes an expressive set of spatio-temporal operators for filter, join with various predicates as well as k nearest neighbor search. A density based clustering operator allows to find groups of similar events.

GeoSpark [1] is an in-memory cluster computing framework for processing large-scale spatial data. GeoSpark is used to load, process, and analyze large-scale spatial data in Apache Spark. It provides a set of out-of-the-box Spatial Resilient Distributed Dataset (SRDD) types (e.g., Point RDD and Polygon RDD) that provide in-house support for geometrical and distance operations. SRDDs provide an Application Programming Interface (API) for Apache Spark programmers to easily develop their spatial analysis programs.

Magellan is a distributed execution engine for geospatial analytics on big data. It uses database techniques like efficient data layout, code generation and query optimization in order to optimize geospatial queries. Developers can write either standard SQL or data frame queries for spatial operations, while the execution engine efficiently takes care of laying data out in memory during query processing. Magellan extends Spark SQL to provide a relational abstraction for geospatial analytics.

Spatial-Spark [14] provides efficient spatial operations dedicated for big spatial data using Apache Spark. It provides two different spatial join operators, namely broadcast spatial join and partitioned spatial join. Broadcast spatial join is designed for joining one big dataset with another small dataset efficiently. Partitioned spatial join is more general for joining two big datasets. Finally, it provides spatial range queries with/whithout indexing.

Another notable system that supports distributed execution of spatial operations is the system Simba [13]. Simba extends the Spark SQL engine across the system stack to support rich spatial queries and analytics through both SQL and DataFrame query interfaces. Due to the fact that indexing of more complex geometries than points (e.g., polygons) has not

[1] http://spatialhadoop.cs.umn.edu/
[2] https://sites.google.com/site/hadoopgis/
[3] https://github.com/Esri/spatial-framework-for-hadoop

yet been implemented and due to its very limited spatial support[4] we decided not to include it in our functionality and performance benchmarking which is presented in Chapter 4.

Jackpine Benchmark [11] is a state-of-the-art benchmark for spatial SQL queries that can support any database that can be accessed via JDBC. Since it is considered a robust, state-of-the-art benchmark platform for spatial databases, our idea is to extend the usages of its operations into distributed spatial systems to assess the performance of individual spatial SQL functions.

Lastly, It is worth mentioning Geographica [5], a benchmark for geospatial data expressed in RDF. This benchmark is used for the evaluation of the new generation of RDF stores supporting the query languages GeoSPARQL and stSPARQL. Following the approach of Jackpine, it defines a micro benchmark and a macro benchmark. The micro benchmark tests primitive spatial functions. The spatial component of a system is checked with queries that use non-topological functions, spatial selections, spatial joins and spatial aggregate functions. In the macro benchmark it is testing the performance of the selected RDF stores in typical application scenarios like reverse geocoding, map search and browsing, and a real-world use case from the Earth Observation domain.

Exareme, implements a JDBC driver, so the Jackpine platform can be used directly with Exareme to run the supported spatial operations. For the rest of the evaluating systems, that do not implement a JDBC driver, we created applications that to run Jackpine's supported micro benchmark operations.

[4] As noted in [13], at Table 1, row 'Geometric object' and at the corresponding notation 'Simba is being extended to support general geometric objects', but until now it supports spatial operations only between points, p. 3, section 2.4 Spatial Operations

Table 1: Indices and partitions supported by each system

System	Indices	Partitions
Exareme	R*Tree	Grid
GeoSpark	R-Tree (used in the experiments) or Quad-Tree	Global Grid using Equalgrid (used in the experiments), Hilbert, R-Tree, Voronoi or Quad-Tree
Stark	R-Tree For k-means an Extended R-Tree (based on JTSplus implementation for KNN queries)	Fixed Grid Partitioned (used in the experiments) and cost based BSP (Binary Space Partitioner)
Magellan	Z-Order Curve (finally treated as a linear or pointer based Quad-Tree)	None
Spatial-Spark	R-Tree	Fixed-Grid Partition (FGP), Sort-Tile Partition (STP), Binary-Split Partition (BSP)

4. IMPLEMENTATION

In this chapter we describe a functionality and performance benchmark as implemented for this diploma thesis. We describe the functionalities of the state-of-the-art distributed systems that support spatial queries and we compare them. Table 2 briefly describes the indexing and partitioning mechanisms supported in these systems, as well as the respective programming languages that they support, which are further explained.

Table 2: Functionality overview

System	Spatial Index	Spatial Partitioning	Language
STARK	R-Tree	Grid	DSL
Geospark	R-Tree, Quad-Tree	Grid	Java API
Magellan	Z-order		SQL
Spatial-Spark	R-Tree	FGP, BSP, STP	Scala
Exareme	R-Tree	Grid	SQL

4.1. Systems

4.1.1. STARK

The STARK framework has been designed for scalable spatio-temporal analytics on Spark, extending it with dedicated data types and operators for spatio-temporal data. In contrast to other similar existing solutions, STARK is the only framework that supports both spatial and spatio-temporal data. STARK is tightly integrated into the Apache Spark API so that users can directly invoke the spatio-temporal operators and their RDDs. This is achieved by creating new data type and operator classes that make use of already existing Spark operations, but also extend internal Spark classes.

Partitioning Apache Spark's partitioners don't exploit spatial (or spatio-temporal) characteristics. Spatial-temporal partitioning means that partitions are not created by taking into consideration the location in space and/or time. Also, if the partitions sizes are not balanced then a single node can do most of the work and the others stay idle. STARK considers both the spatial component for creating and balancing partitions. Two spatial partitioners are implementing Spark's Partitioner interface so that they can be used to spatially partition an RDD with the RDD's *partitionBy* method.

The *Grid Partitioner* is a fixed grid partitioner where it divides data space into a number of intervals per dimension resulting in a grid of rectangular cells (partitions) with equal dimensions. Because all cells have equal size, some of them may contain more data items than others. To overcome this problem, the *Cost-Based Binary Space Partitioner*, based on [9] divides the space into two partitions with equal cost (number of contained items). When the items of one partition are exceeding a threshold, it is recursively divided

into two partitions of equal cost. This way, large regions with only a few items will belong to the same partition, while dense regions are split into multiple partitions.

Indexing STARK uses an R-Tree (STR-Tree) implementation for indexing the contents of a partition. STARK has three indexing modes:

- No Indexing. The partitions are not indexed and all items within a partition have to be evaluated with the respective predicate function.
- Live Indexing. While processing a partition for evaluating a predicate, the contents of that partition are indexed on-the-fly while the query is evaluated.
- Persistent Indexing. STARK allows the materialization of the R-Tree index to disk/HDFS using Spark's method to save binary objects. That way, a persisted index can be used eliminating the cost of indexing data on-the-fly in the execution time of a query.

Query Language STARK provides an integrated DSL (domain specific language) for spatio-temporal query processing that seamlessly integrates into any (Scala) Spark program. Spatial Joins and filters can be called directly as transformations on standard RDDs. Additionally, it allows defining custom distance functions and predicates for its operators. More specifically, it supports the following operators: `intersect`, `contains` and `containedBy`. It does not support SQL.

4.1.2. GeoSpark

GeoSpark is a Java implementation and it consists of three layers: (i) the Apache Spark layer that provides the basic Apache Spark functionality, (ii) the Spatial Resilient Distributed Dataset (SRDD) layer that offers an extension to Apache Spark's RDD, and (iii) the Spatial Query Processing layer that is based on the SRDD and supports spatial queries (range and join) for large-scale spatial datasets.

The SRDD layer supports five different RDD types such as PointRDD, LineStringRDD, CircleRDD, RectangleRDD and PolygonRDD to handle geometrical objects and it also supports spatial operations on these RDD types.

Partitioning GeoSpark partitions all loaded Spatial RDDs by creating one global grid file for data partitioning. The Spatial RDDs are using the Apache Spark layer by extending Spark's partitioner. Global grids are low cost in either file-size or data partitioning but constructing a global grid file demands multiple coordinate sorting on the same datasets. It supports several partitioning techniques such as Equalgrid (used in the experiments), Hilbert, R-Tree, Voronoi and Quad-Tree.

Indexing Spatial indexes like R-Tree (STR-Tree that is used in the experiments) or Quad-Tree are provided in the Spatial Query Processing layer. The Sort-Tile-Recursive (STR) algorithm is a simple and efficient bulk-loading method for spatial or multidimensional data management using R-tree [10]. Finally, GeoSpark's index can be persisted either in memory or in disk for later from the same program.

Query Language As mentioned above, GeoSpark can be used via its Java API. Users have to create SRDD objects and pass them into their JavaRDD base. For each operation, an object of the supported operations has to be created and this will return a List, a JavaPairRDD or a JavaRDD object. The supported spatial operations are Spatial Join Query, Spatial KNN Query and Spatial Range Query. The following operators are supported as spatial join conditions: `Overlap`, `Inside`, and `Disjoint`. Construct operators such as `MinimumBoundingRectangle` and group operators such as `Union` are also supported. SQL is not supported.

4.1.3. Magellan

Magellan is built on top of Apache Spark to create a distributed engine for geospatial analytics on big data and it extends Spark SQL to support geospatial queries. Spatial information can be given as input in the following formats: GeoJSON, OSM-XML and WKT. The following geomtry types are supported: Point, LineString, Polygon, MultiPoint, MultiPolygon. Magellan also supports the following topological functions that can be used in SQL queries : `Intersects`, `Contains` and `Within`.

Partitioning Magellan does not perform spatial partitioning. Instead, it can use Spark's partitioner in order to provide an RDD partitioning based on the data provided.

Indexing As spatial index it uses a Z-Order Curve, which is finally treated as a linear or pointer based Quad-Tree. Given a column of shapes, the values are indexed using a geohash function. This produces a new column which is a list of Z-Order Curves of a specific precision that cover the respective shape. While loading a spatial file (in GeoJSON, Shapefile or SM-XML format), spatial data gets automatically indexed.

Query Language Magellan extends Spark SQL with spatial support. In that way it inherently supports relational spatial joins. However, these joins are not handled as spatial joins by default, unless a spatial join rule is injected. Otherwise, the join will be evaluated as a Cartesian Join followed by a predicate. [1]

[1] As mentioned in https://github.com/harsha2010/magellan paragraph Spatial Joins

4.1.4. Spatial-Spark

Spatial Spark was one of the early systems that extended Spark with spatial support. It provides a large-scale spatial join query processing on two leading in memory Big Data systems, namely Apache Spark and Cloudera Impala [14]. To achieve this, they focus on data processing on parallel hardware like multi-core CPUs and GPUs. Spatial-Spark implements two different kinds of spatial joins: Broadcast Spatial Join and Partitioned Spatial Join. With broadcast join the object in the right relation is read into memory and distributed to all workers. If the relation is too big to fit in memory then the Partitioning Join can be used instead.

Partitioning Spatial-Spark with Spatial Partitioning Join uses Fixed-Grid Partition (FGP), Binary-Split Partition (BSP) and Sort-Tile Partition (STP). As mentioned above, when a dataset is too big to fit into the main memory then a partitioner is used to distribute the data in different nodes. BSP and STP partitioners are using R-Tree to distribute and broadcast data while FGP is using the fixed-grid approach. In the Broadcast Spatial Join, when data does not fit in memory, the default Spark partitioner is employed in order to distribute the data, without the ability to provide spatial partitioning.

Indexing Spatial-Spark uses an R-Tree index in Broadcast Spatial Join for spatial operations. Indexing is also used during spatial range queries but not in Partitioned Spatial Join queries.

Query Language Spatial Spark can be used via the command line and run single operations or as a library in Scala programs. It currently supports the following operators: intersect, contains, within, within distance, overlaps and nearest distance. Spatial-Spark does not support SQL queries.

4.1.5. Exareme

Exareme [3] is a system for elastic large-scale data processing on the cloud that follows a more general paradigm. It operates as a paralleled RDBMS that uses a master-worker architecture and each parallel node is running an instance of Exareme. The core component of Exareme is a data flow system named MadIS. MadIS is based on an APSW wrapper of SQLite[2] to process queries over data that are stored (or can be virtually seen) as tables. For the spatial extension of Exareme, we enabled the spatial extension of SQLite, named Spatialite[3].

From a user's point of view, the system is used as a traditional database system: create/drop tables or indexes, import external data, issue queries, while its language is based

[2]http://www.sqlite.org/

[3]https://www.gaia-gis.it/fossil/libspatialite/index

on SQL to express both intra-worker and inter-worker dataflows. Exareme uses UDFs and an inverted syntax to easily express local pipelines and complex computations. Inter-worker dataflows are described with simple parallelism primitives.

Partitioning Data are inserted in form of database table inputs. A table is defined as a set of partitions and a partition is defined as a set of records having a particular property. Partitions are created during the data loading into the system. For non-spatial data, the declaration of the table creation includes the number of partitions it will be split into with respect to a specific column. However, when joining two tables that are partitioned into different number of partition on the same property, Exareme performs re-partitioning: One of two tables will be re-partitioned on-the-fly so that a direct join can be performed or the join is implemented using Cartesian product.

For spatial data, geometries are partitioned using a grid. Partitioning geometries is a more challenging task than partitioning numeric values, as geometries are typically multi-dimensional values. The overhead of re-partitioning is bigger when geometries are involved as geometric computations care typically more costly. Partitions that overlap with a cell of a grid are contained in the same partition that corresponds to this cell. Every partition (e.g., grid cell) is assigned to a different worker to facilitate the parallel execution of a spatial query. Each worker fetches the partitions needed for the execution and caches them to its local disk for subsequent usage.

Indexing Exareme is using R*Tree for spatial indexing which is provided by SpatiaLite. As we explained above, every cell of the grid that the spatial data are partitioned corresponds to a different partition. For every partition, an R-Tree index is constructed. When a spatial join needs to be evaluated, every worker is assigned to process locally a different cell of the grid. The local R-tree index that is constructed for the respective raster cell may be used if the optimizer decides so.

Query Language The system offers a declarative language which is based on SQL with user-defined functions (UDFs) extended with parallelism primitives and an inverted syntax to easily express data pipelines. Exareme adopts the relational data model and extends it with UDFs that can be implemented by the user as row, aggregate, or virtual table operators and can be used embedded in SQL queries. The query language supported by Exareme is ExaQL, an extension of SQL-92 with user-defined functions (UDFs). Queries can also be issued to Exareme using a lower-level language, named ExaDFL. ExaDFL is a dataflow language that describes DAGs and it's based on SQL extended with UDFs and data parallelism primitives. The spatial component of Exareme inherits the spatial functionalities of Spatialite.

The goal of the micro benchmark is to test the basic topological relationships and spatial analysis functions. A topological relationship describes how two spatial objects relate to each other in terms of topological constraints. The geometric objects could be point, line or polygon.

Table 3: Topological relations, all pair joins

Operation	Description	Query
Equals	Polygon equals polygon	Find the polygons that are spatially equal to other polygons in `arealm_merge` table
Equals	Point equals point	Find the points that are spatially equal to other points in `pointlm_merge` table
Disjoint	Polygon disjoint polygon	Find the polygons that are spatially disjoint from other polygons in `arealm_merge` table
Intersects	Line intersects polygon	Find the lines in `edges_merge` table that intersect polygons in `arealm_merge` table
Intersects	Point intersects polygon	Find the points in `pointlm_merge` table that intersect polygons in `arealm_merge` table
Intersects	Point intersects line	Find the points in `pointlm_merge` table that intersect lines in `edges_merge` table
Intersects	Line intersects line	Find the lines that intersect lines in `edges_merge` table
Touches	Polygon touches polygon	Find the polygons that touch polygons in `arealm_merge` table
Touches	Line touches polygon	Find the lines in `edges_merge` table that touch polygons in `arealm_merge` table
Crosses	Line crosses line	Find first 5 lines that crosses other lines in `edges_merge` table
Crosses	Line crosses polygon	Find the lines in `edges_merge` table that cross polygons in `arealm_merge` table
Overlaps	Polygon overlaps polygon	Find the polygons that overlap other polygons in `arealm_merge` table
Overlaps	Line overlaps polygon	Find the lines in `edges_merge` that overlap polygons in `arealm_merge` table
Within	Polygon within polygon	Find the polygons that are within other polygons in `arealm_merge` table
Within	Line within polygon	Find the lines in `edges_merge` table that are inside the polygons in `arealm_merge` table
Within	Point within polygon	Find the points in `pointlm_merge` table that are inside the polygons in `arealm_merge` table
Contains	Polygon contains polygon	Find the polygons that contain other polygons in `arealm_merge` table
Contains	Polygon contains point	Find the polygons in `arealm_merge` that contain points in `pointlm_merge` table

Spatial analysis functions are analytic operations to determine the spatial properties of interest.

The queries included in the micro benchmark should provide complete, yet minimal, cov-

Table 4: Datasets used for micro Benchmark

Dataset name	Geometry	Cardinality	Data file size (Bytes)
edges_merge	line	5895060	2,034,869,136
pointlm_merge	point	13441	821,049
arealm_merge	polygon	5963	3,756,243

Table 5: Micro Benchmark supported operations per system (Polygons)

	Exareme	GeoSpark	Stark	Magellan	Spatial-Spark
Polygon Contains Point	Y	Y	Y	Y	Y
Polygon Contains Polygon	Y	Y	Y		Y
Polygon Disjoint Polygon	Y				
Polygon Equals Polygon	Y				
Polygon Overlaps Polygon	Y				Y
Polygon Touches Polygon	Y				
Polygon Within Polygon	Y				Y
Line Crosses Polygon	Y				
Line Crosses Line	Y				
Line Intersects Polygon	Y		Y		Y
Line Intersects Line	Y		Y		Y
Line Overlaps Polygon	Y	Y			Y
Line Touches Polygon	Y				
Line Within Polygon	Y				Y
Point Equals Point	Y				
Point Intersects Line	Y		Y		Y
Point Intersects Polygon	Y		Y	Y	Y
Point Within Polygon	Y			Y	Y

erage of topological relations.

Following Jackpine's approach, we are using the Dimensionally Extended Nine-Intersection Model (DE-9IM) which proposes the relationships: Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains and Overlaps. The DE-9IM has been adopted by the Open Geospatial Consortium.

For our evaluation we are using a subset of Jackpine's micro Benchmark topological relationships among polygon, line and points, those concerning topological relations and pair joins as shown in table 3. Table 4 shows the data models that are used for our evaluation.

The supported operations per system are shown in table 5.

4.2. Performance benchmark

4.2.1. Experimental set up

For our experiments we used the Okeanos Cloud Infrastructure[4]. The experiments were executed using a cluster of 3 VMs, consisted by 8 Cores CPU, 8 GB RAM and 60 GB Storage capacity each. We installed Ubuntu 16.04.4 LTS to VMs and configured the cluster to provide HDFS for storage (Hadoop 2.7.3), and Apache Spark 2.2.0.

The same cluster was used also for the evaluation of Exareme but without the HDFS and Apache Spark. For the spatial support we installed SpatiaLite 4.3.0a, based on SQLite 3.11.0.

Apache Spark's mechanisms are responsible for the distribution of the resources and partitioning of data. We set the amount of memory for the driver process (`spark.driver.memory`) to 10 GB and the Default timeout for all network interactions (`spark.network.timeout`) to 1000 seconds. Finally, we set the limit of total size of serialised results of all partitions for each Spark action (`spark.driver.maxResultSize`) to 5 GB. This value was set in order to protect the driver from out-of-memory errors.

In Exareme, each worker is occupying a VM and consumes its resources to process the operations. The configuration and the installation was ready out-of-the-box.

4.2.2. Evaluation results

The results of the benchmark queries are described in this section. In each benchmark we run experiments 4 times, considering the first as the warmup execution and measuring the average execution time for the last 3 runs. Each benchmark is run for all systems multiple times but with different configurations. We used three partition configurations (No partitions, 8 or 16) and three Core configurations (8, 16 or 24 Cores) per run. The scenarios are comprised of 18 pairwise spatial join queries among polygon, line and point objects and the results are shown categorised by these configurations.

For the experiments we used the following indexing and partitioning techniques. For STARK we used the live index with R-Tree and the Spatial Grid Partitioner to partition the data. In GeoSpark we used the R-Tree index and the Equalgrid partitioner. For Magellan we used the default indexer of the system (Z-Order Curve). There is no reference for partitioning in Magellan and so we assume that it uses the Spark's Partitioner. For Spatial-Spark we define two cases. One where we use the Broadcast Spatial Join with R-Tree indexer and the other, where we use Partitioned Spatial Join. On the first there is no spatial partition support and therefore we use Spark's partitioner. On the latter, there is no indexer but it supports spatial partitioning. There we use the Fixed Grid Partitioner to partition the datasets. Finally, in Exareme we used Grid partitioning and R-Tree indices.

[4] https://www.okeanos.grnet.gr/

The results of the experiments are also available online[5].

Among the experiments, we consider as the most informative for the systems, the results of queries that involve spatial joins with complex geometries, such as lines and examine how the systems in comparison scale in these queries.

This is interesting because, in the respective experiment of the Jackpine benchmark, the dataset with line objects is too big to fit in memory, so we wanted to examine the behaviour of the Spark-based systems in this realistic scenario in comparison to Exareme. In such cases, systems like Spatial-Spark and GeoSpark needed to be configured differently than others.

Since Broadcast Spatial Join doesn't support spatial partitioning, the partitions are based on Spark's default partitioner. In the experiments with Partitioned Spatial Join (no indexer here), the Fixed Grid partitioner is using the same technique when needed for large datasets.

GeoSpark supports persistent index, so, we enabled persistence to Disk for line objects. This was also required in non indexed queries where we enabled persistence for partitioning as well, because we were receiving error messages for insufficient memory during tests otherwise.

In the following figures we provide diagrams of the results for experiments related to the scalability of the geometric objects. For each Spark system two diagrams are provided. A diagram that compares the referenced system with Exareme and a diagram that shows the geometric scalability with and without indexer.

Exareme has been tested only with the spatial indexer. In the diagrams, Exareme results are shown with the star pointer and the results of the Spark based system are shown with the triangle pointer.

Correspondingly and in order to provide an easy to look comparison between the two diagrams, the results with index are shown with the triangle pointer and the results without index are shown with the star pointer. The colour coding of the two diagrams have been kept the same when possible.

The values used to produce the results are referring to experiments with datasets in 8 partitions. This number has been selected because all systems give useful results that can be compared to each other.

In figures 1, 2, 3, 4, 5, 6, 7 and 8 we provide diagrams of the results for experiments related to line objects. Some diagrams may refer to point scalability but we mentioned them for line objects as well.

[5]https://github.com/kgiann78/GeoSpatial-Distributed-Systems

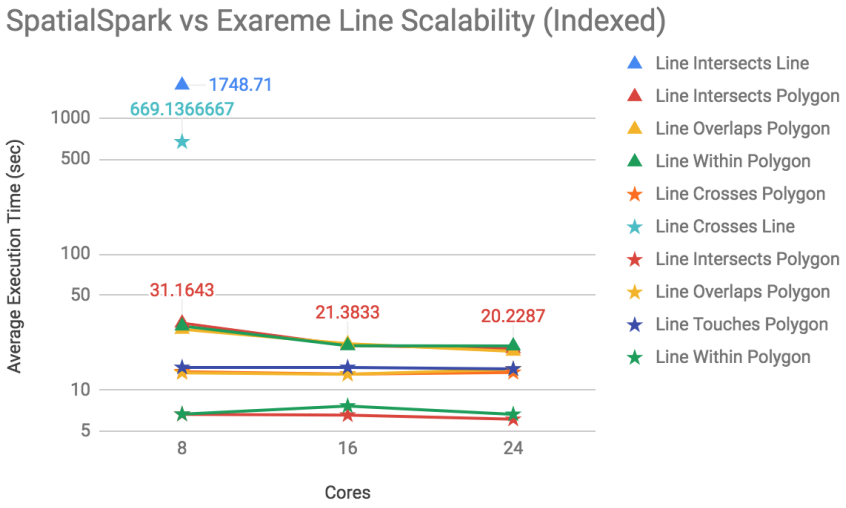

Figure 1: SpatialSpark vs Exareme Line Scalability

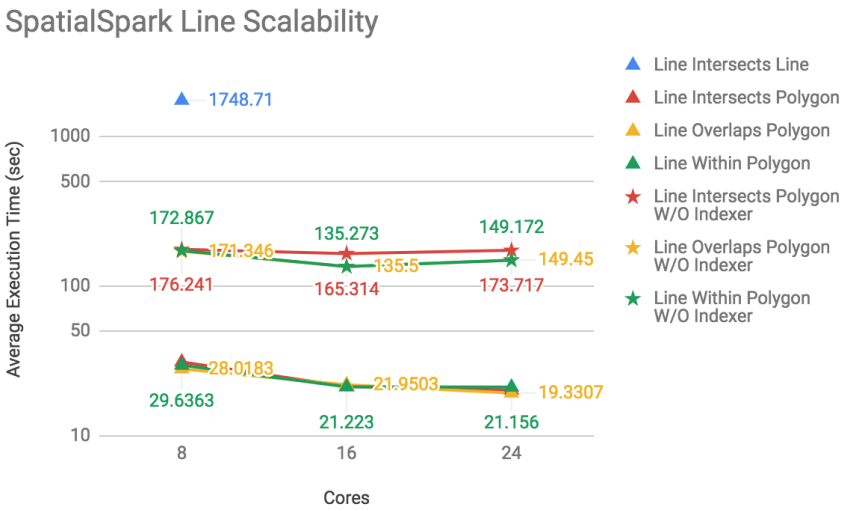

Figure 2: SpatialSpark Line Scalability

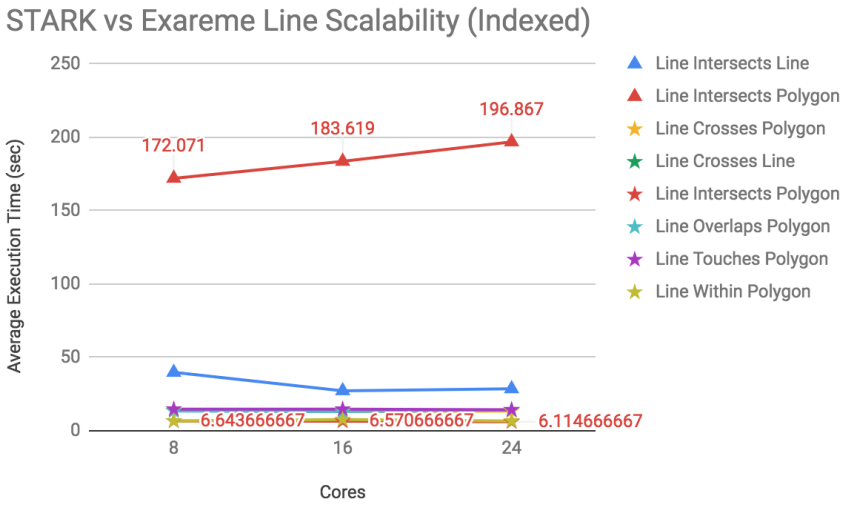

Figure 3: STARK vs Exareme Line scalability

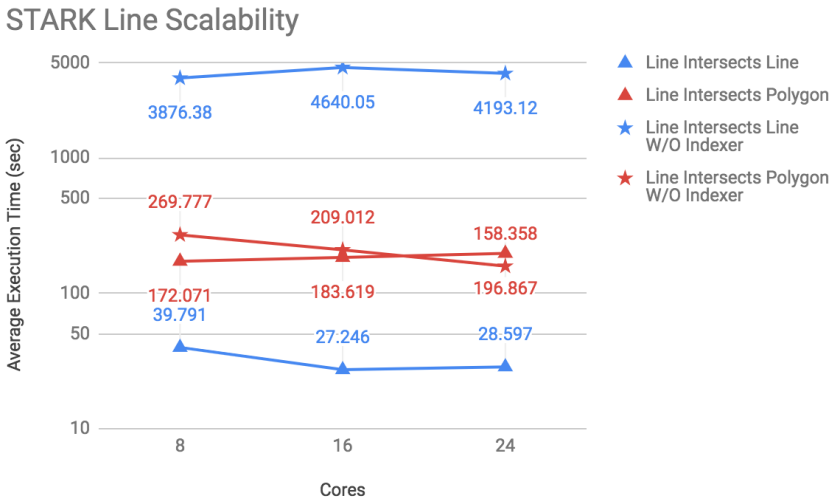

Figure 4: STARK Line scalability

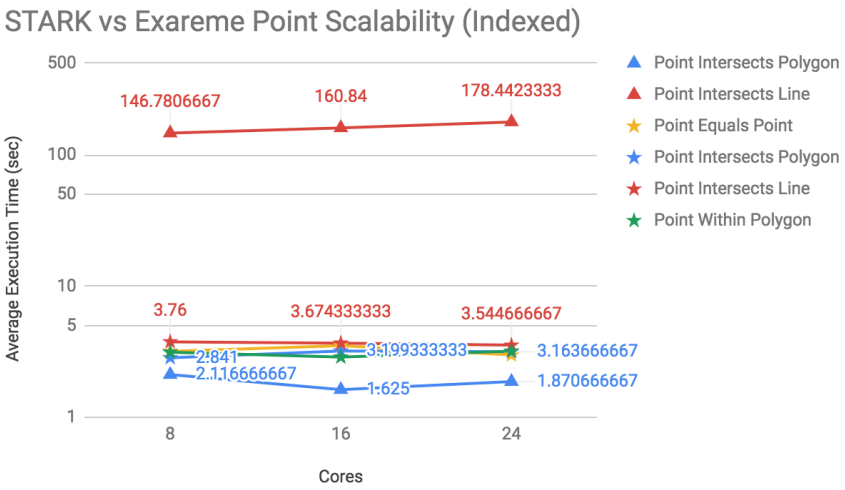

Figure 5: STARK vs Exareme Point scalability

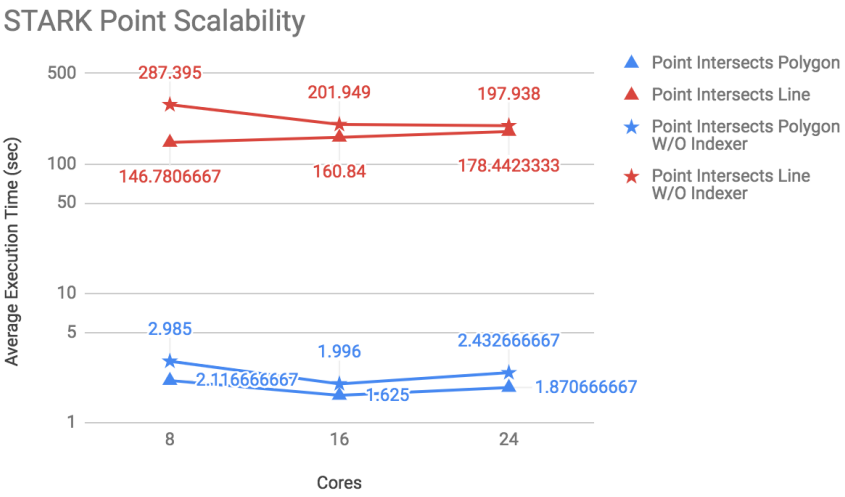

Figure 6: STARK Point scalability

In STARK we are using live index, which means that in each experiment it creates on the fly an index. The content of a partition is first put into an R-tree and then, this index is queried using the query object. The elements of the R-Tree have to be checked again to see if they really match the query object. For small datasets, this performs better but for larger datasets it creates some latency due to the number of elements and the network communication as it can be seen in figures 4 and 6.

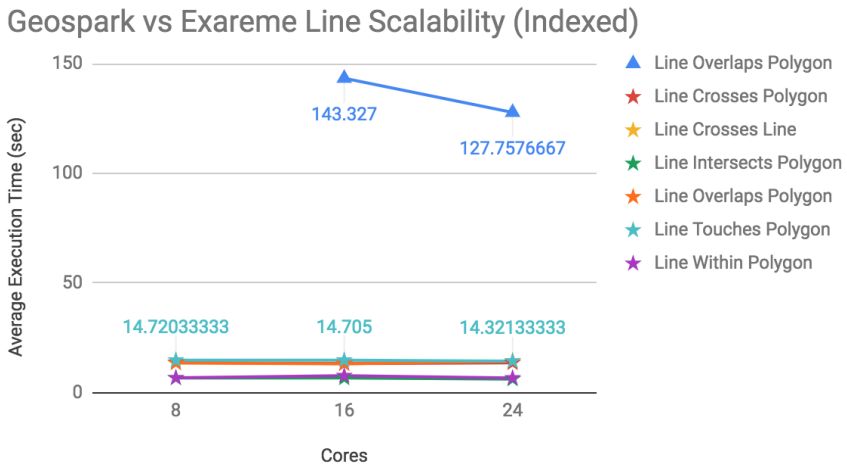

Figure 7: Geospark vs Exareme Line Scalability

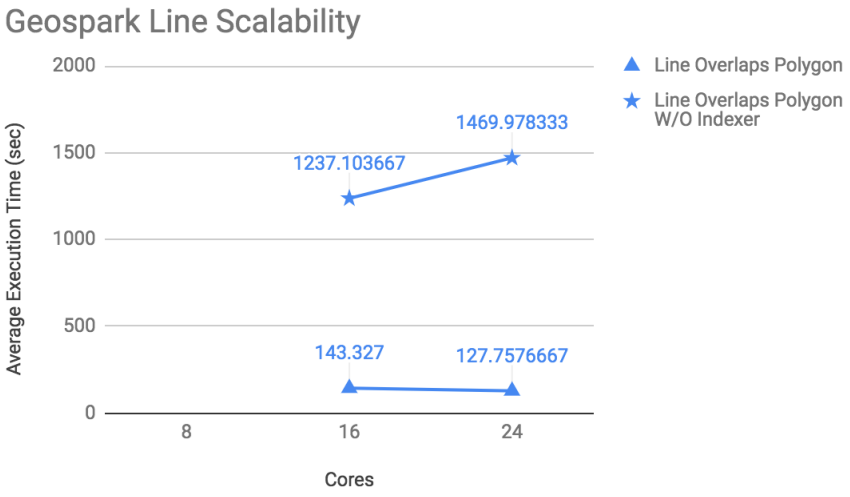

Figure 8: Geospark Line Scalability

GeoSpark, is using an R-Tree index per each partition, after data are partitioned. This is supposed to be fast but in our experiments it increases the communication between different partitions. Especially, for large datasets, it has a significant performance impact (Line overlaps polygon fig. 8).

Magellan, provides a clever way to enhance Spark SQL with spatial support but the lack

of spatial partitioning and the limited number of operations didn't give much to the comparisons.

In figures 9, 10, 11, 12, 13, 14, 15 and 16 we present the polygon scalability among systems.

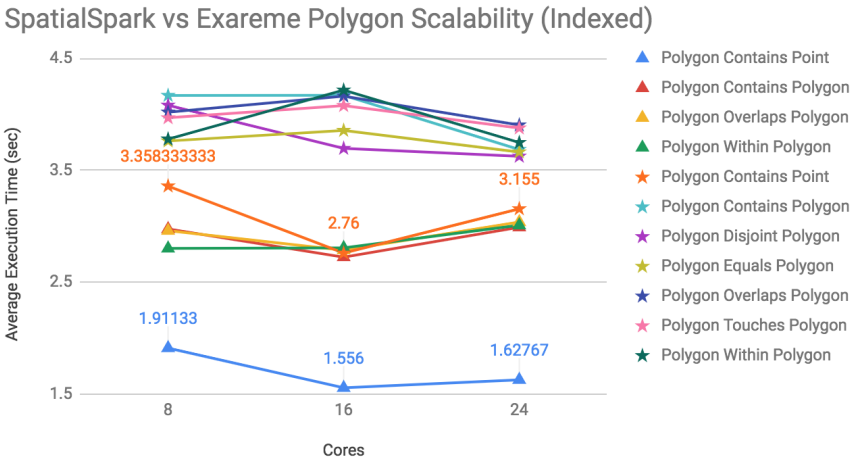

Figure 9: SpatialSpark vs Exareme Polygon Scalability

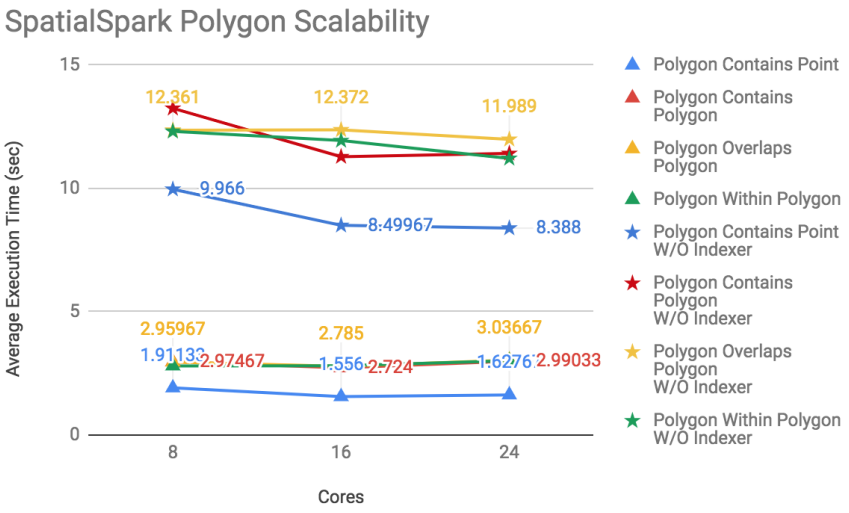

Figure 10: SpatialSpark Polygon Scalability

Finally, in figures 5, 6, 17, 18, 19 and 20 we provide diagrams of the results for experiments related to the scalability of the point geometries among systems.

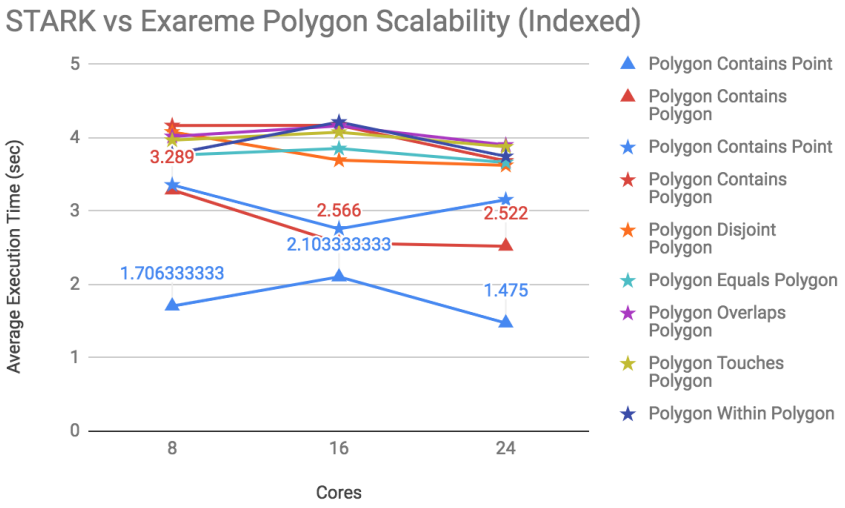

Figure 11: STARK vs Exareme Polygon scalability

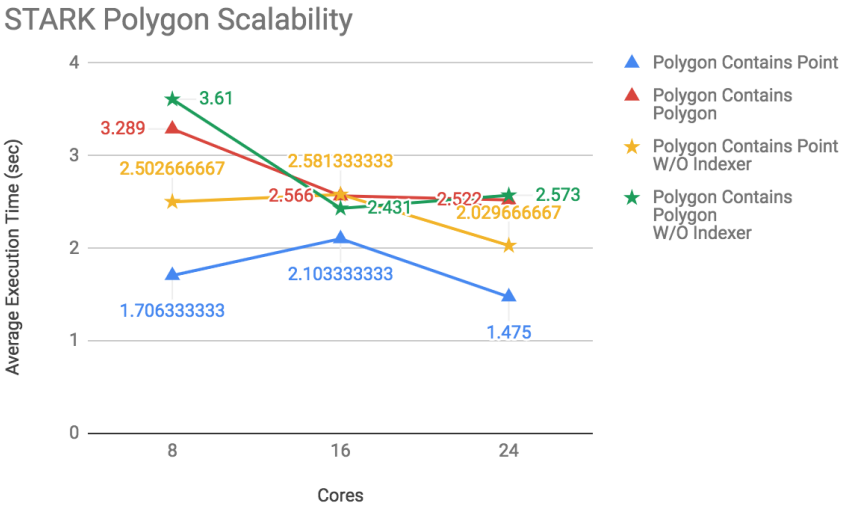

Figure 12: STARK Polygon scalability

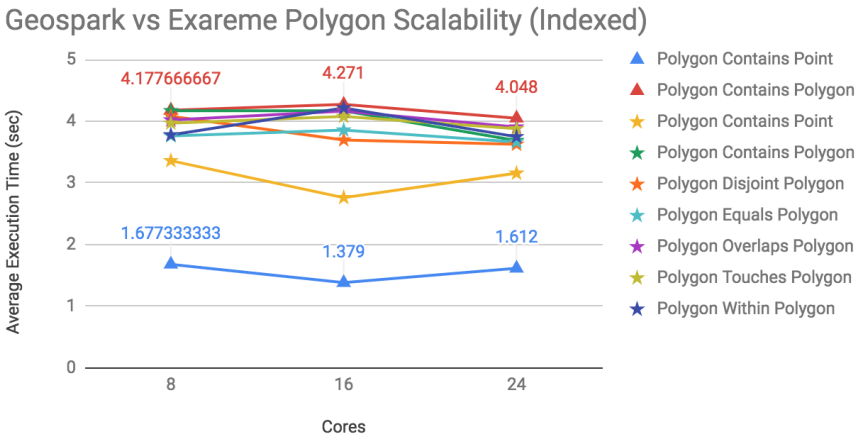

Figure 13: Geospark vs Exareme Polygon Scalability

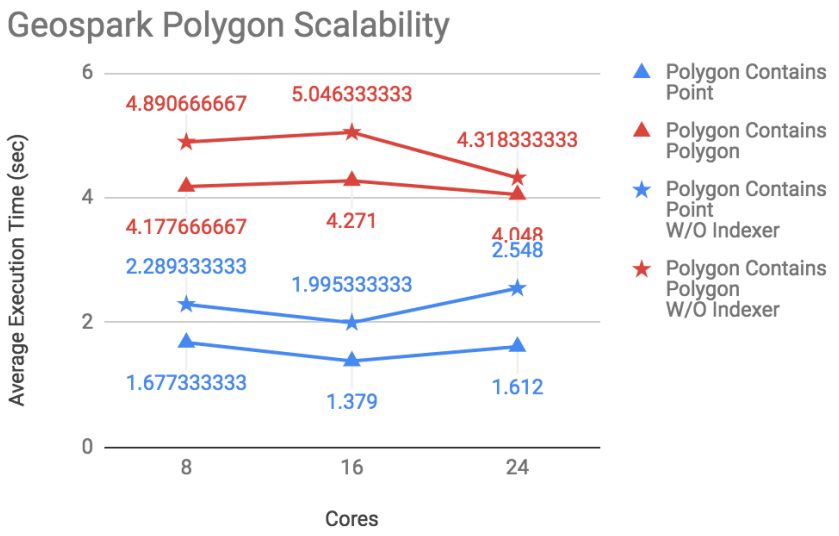

Figure 14: Geospark Polygon Scalability

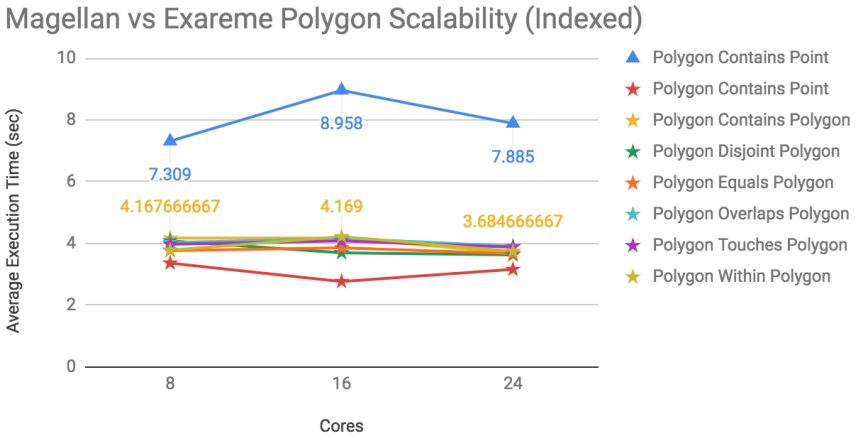

Figure 15: Magellan vs Exareme Polygon scalability

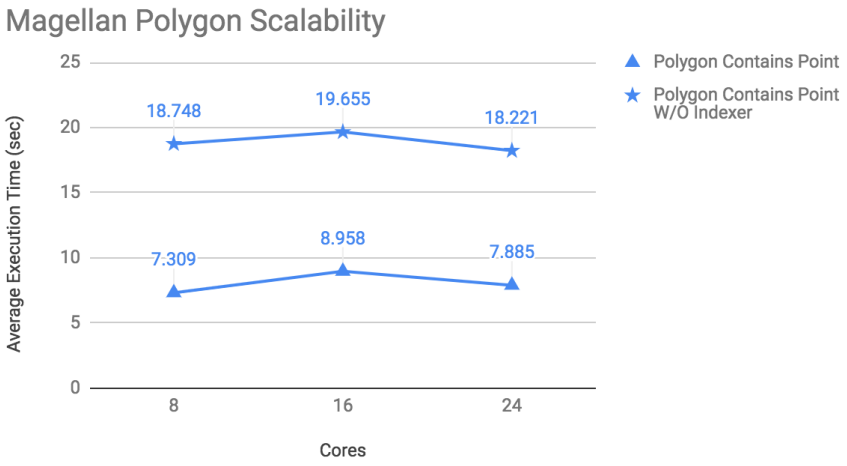

Figure 16: Magellan Polygon scalability

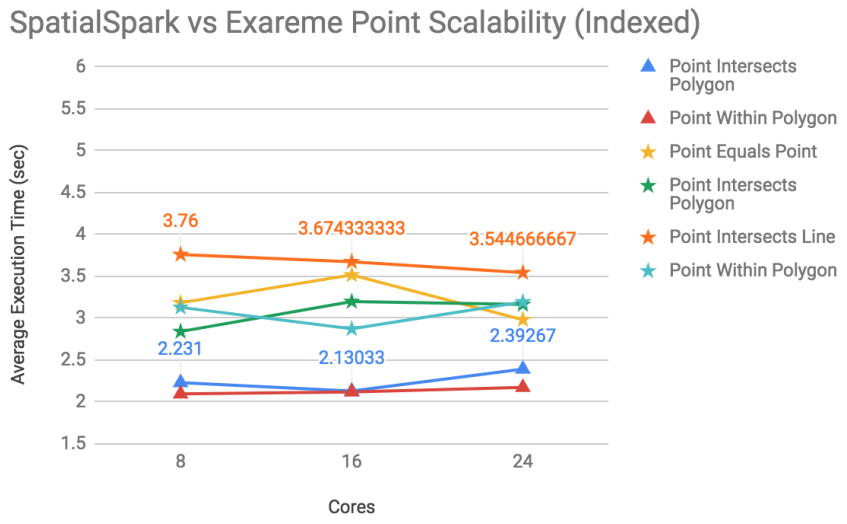

Figure 17: SpatialSpark vs Exareme Point Scalability

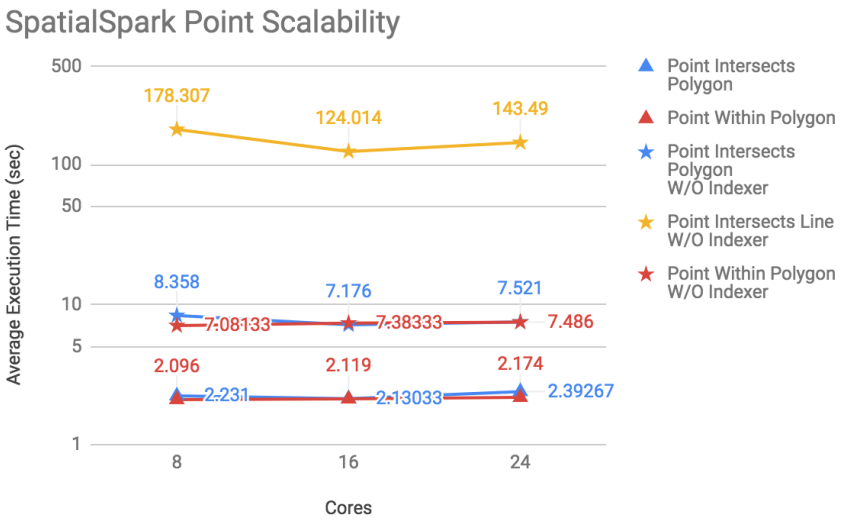

Figure 18: SpatialSpark Point Scalability

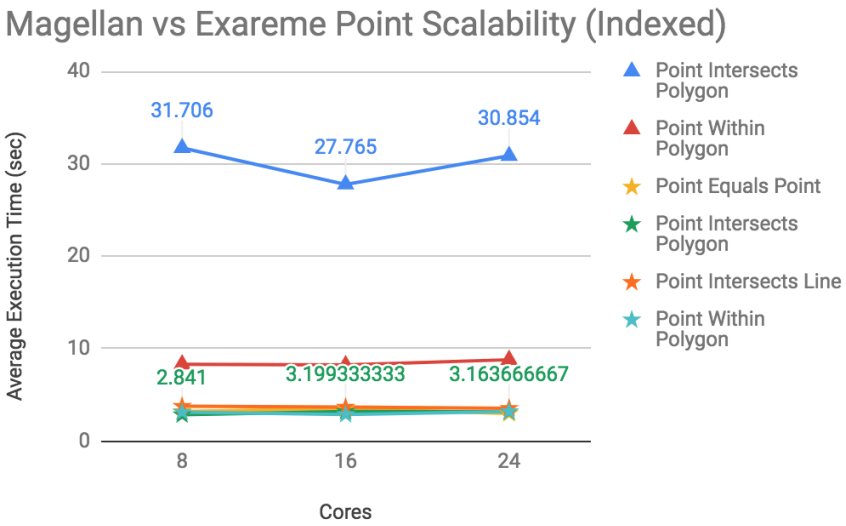

Figure 19: Magellan vs Exareme Point Scalability

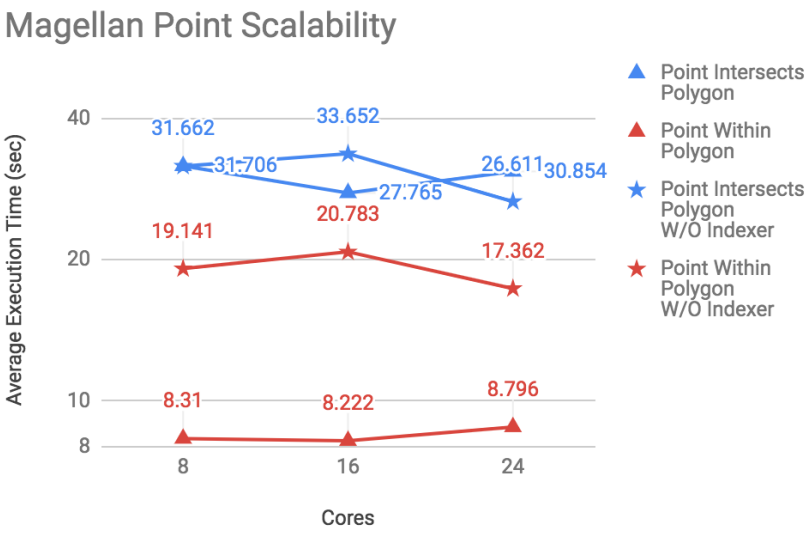

Figure 20: Magellan Point Scalability

In general, each system performed differently in various tests. Considering the number of operations, Exareme implements almost all of them[6] and the results are within a steady range for all different configurations and in most of the experiments. When the datasets are small, SpatialSpark with Spatial Broadcast Join even if it uses Spark's Partitioner has performed better than others. Considering the number of operations, SpatialSpark comes right after Exareme. Unfortunatelly, it performed poorly in cases where a large dataset was member of a spatial operation. On the other hand, STARK implements less operations than Exareme and SpatialSpark but performs better in many cases. Of course, it should be taken into consideration that the number of partitions is larger than other systems with the same nominal number, due to the grid partitioning. Geospark and Magellan implement the less operations, without many differences to Exareme or other systems.

In the following sections we present the evaluation results from all micro benchmark operations' results in comparing tables and figures. The tables in these sections are based in the experiments that are grouped by partitions and parted by the system configuration. Three different configurations per system provide multiple different results per partition. The grouping on the partition number has been decided due to the lack of some systems to support multiple partitions or use different partitioning than others (i.e. some experiments run on SpatialSpark in different partitions than other systems).

Additionally, there are experiments where some systems don't return results either in some or all configurations. The partial results are shown but in case of complete lack of results, nothing is shown in the table.

In sections 4.2.2.1 to 4.2.2.7 we provide the results of the benchmarks for polygon geometries interacting to other polygons or points. No line geometries were used in these benchmarks as the right members of the relationships examined.

Next, in sections 4.2.2.8 to 4.2.2.14 we provide the results for line geometries. In scenario 'Line Crosses Line' (section 4.2.2.8) only Exareme provided partial results. The reason that some benchmarks didn't return anything must have to do with the miscommunication among the nodes of the cluster.

In scenario 'Line Intersects Line' (section 4.2.2.10) the line geometries are both the left and right member of the relationship. For this benchmark we had to change the setup for Spatial Spark. Because the line geometry as the right member of the relationship doesn't fit in memory for the Broadcast Spatial Join we needed to a different setup for the system. So we used instead a non spatial partitioning of namely 256 and 512 partitions in order to achieve some partial results. The reason that we used **non spatial partitioning** is because Broadcast Spatial Join doesn't support spatial partitioning. On the other hand, Spatial Spark without indexer, Partitioned Spatial Join, that uses spatial partitioning didn't return any results even with a setup like above (256 and 512 partitions).

In the same scenario, although Exareme supports operation 'Line Intersects Line', it didn't return any result after a long period of time (more than 5 hours) and the benchmark had to be terminated.

[6] Except 'Line Intersects Line' which didn't finish

Finally, in sections 4.2.2.15 to 4.2.2.18 we provide the results for point geometries. In the ‘Point Intersects Line‘ scenario (section 4.2.2.16) the Broadcast Spatial Join of Spatial Spark didn’t return any results (since line geometry was the right member of the relationship) even with a different setup for partitioning like above.

On the other hand, Spatial Spark with Partitioned Spatial Join returned partial results for the a setup like the above (64 and 256 partitions).

4.2.2.1. Polygon Contains Point

Table 6: Comparing systems with indexers, no partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	GeoSpark Indexed	Magellan Indexed
8	3.020333333	7.52133	0.9113333333	3.626666667	7.309
16	2.799	5.073	1.055333333	3.097	8.958
24	3.140666667	7.56633	4.225	3.024666667	7.885

Table 7: Comparing systems with indexers, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	GeoSpark Indexed
8	3.358333333	1.91133	1.706333333	1.677333333
16	2.76	1.556	2.103333333	1.379
24	3.155	1.62767	1.475	1.612

Table 8: Comparing systems with indexers, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	GeoSpark Indexed
8	4.935666667	1.56333	3.93	1.133
16	4.661666667	1.88267	3.216	1.399333333
24	4.161333333	1.465	2.799666667	1.524666667

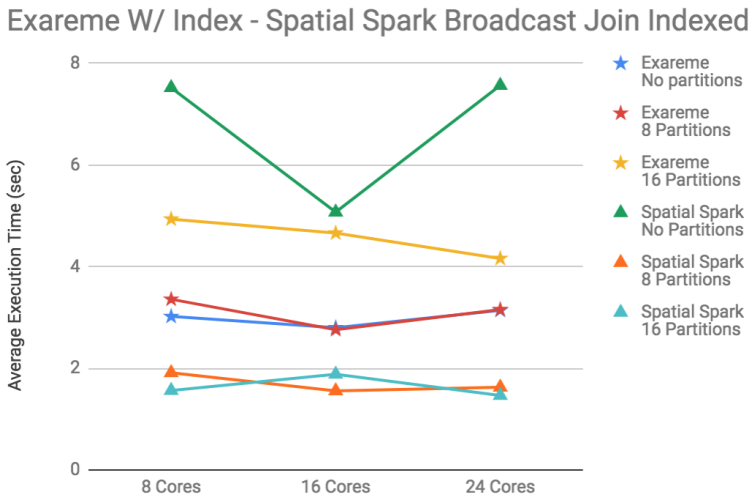

Figure 21: Polygon Contains Point, Comparing Exareme to Spatial Spark

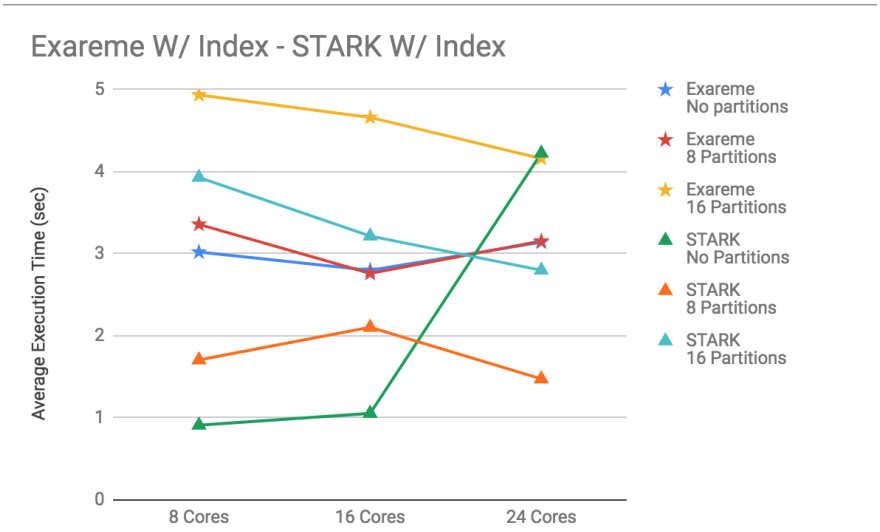

Figure 22: Polygon Contains Point, Comparing Exareme to STARK

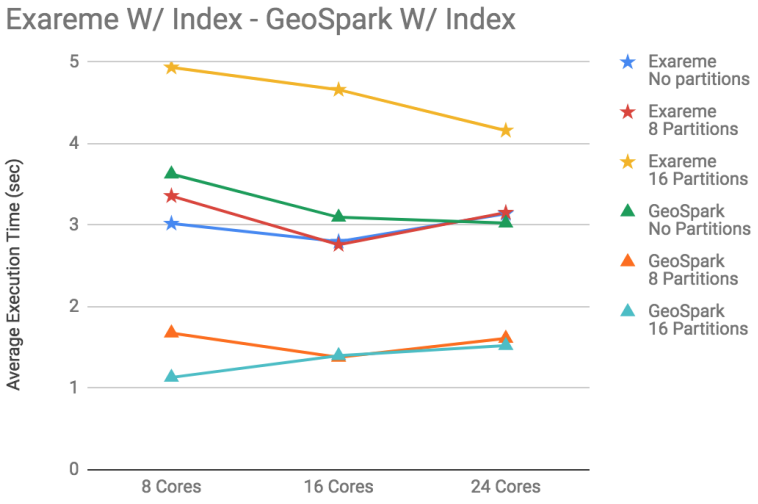

Figure 23: Polygon Contains Point, Comparing Exareme to GeoSpark

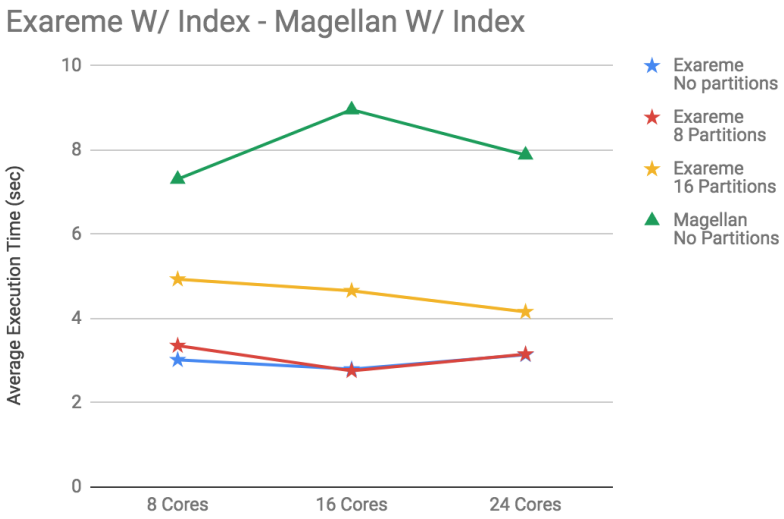

Figure 24: Polygon Contains Point, Comparing Exareme to Magellan

Table 9: Comparing systems without indexers, no partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	GeoSpark w/o Indexer	Magellan w/o Indexer
8	21.22033333	7.448	5.611333333	18.748
16	16.19333333	7.834	4.978666667	19.655
24	18.03566667	6.451333333	5.341666667	18.221

Table 10: Comparing systems without indexers, 8 partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	GeoSpark w/o Indexer
8	9.966	2.502666667	2.289333333
16	8.49967	2.581333333	1.995333333
24	8.388	2.029666667	2.548

Table 11: Comparing systems without indexers, 16 partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	GeoSpark w/o Indexer
8	5.733	4.453333333	1.462333333
16	5.276	4.663333333	1.468333333
24	4.99467	3.623333333	1.566666667

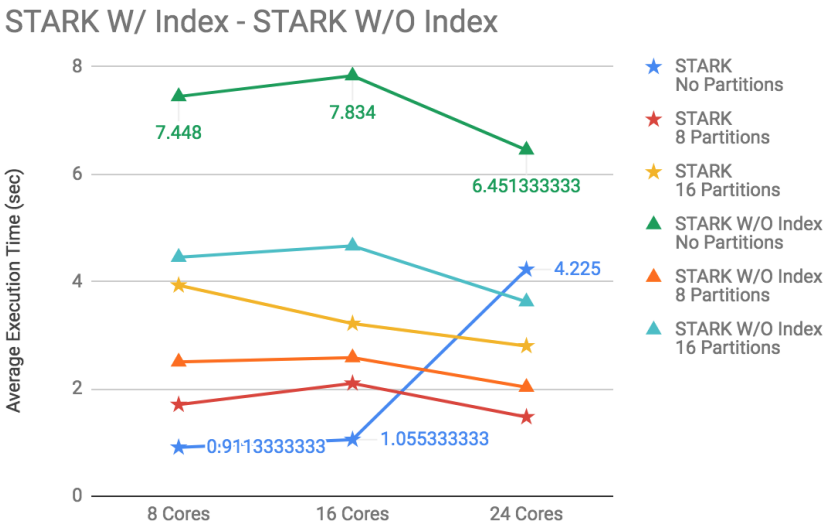

Figure 25: Polygon Contains Point, Comparing STARK W/ and W/O indexers

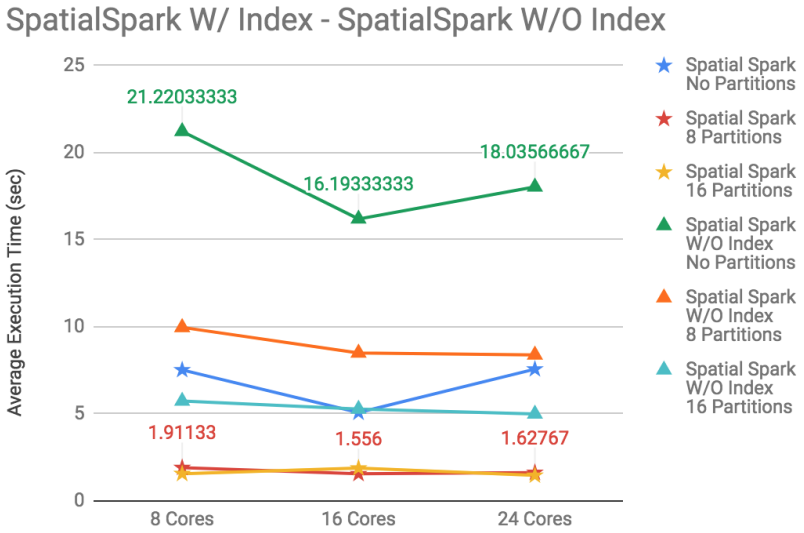

Figure 26: Polygon Contains Point, Comparing Spatial Spark W/ and W/O indexers

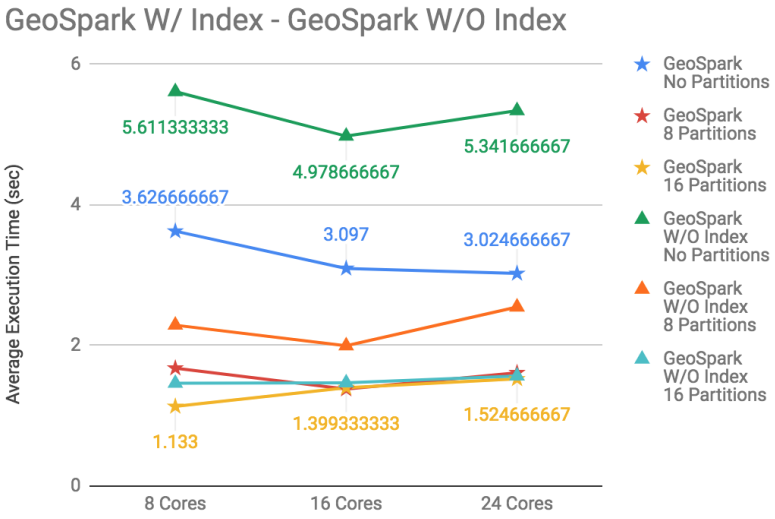

Figure 27: Polygon Contains Point, Comparing GeoSpark W/ and W/O indexers

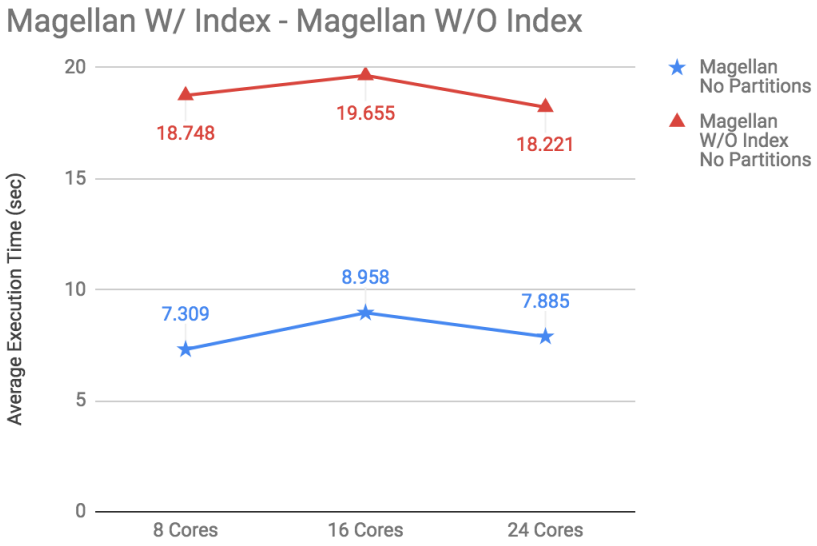

Figure 28: Polygon Contains Point, Comparing Magellan W/ and W/O indexers

4.2.2.2. Polygon Contains Polygon

Table 12: Comparing systems with indexers, no partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	GeoSpark Indexed
8	5.866333333	9.21467	5.167	5.28467
16	6.15	8.54933	6.407	4.97733
24	6.185333333	11.186	5.946	4.909

Table 13: Comparing systems with indexers, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	GeoSpark Indexed
8	4.167666667	2.97467	3.289	4.177666667
16	4.169	2.724	2.566	4.271
24	3.684666667	2.99033	2.522	4.048

Table 14: Comparing systems with indexers, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	GeoSpark Indexed
8	5.445666667	2.796	4.853	3.24067
16	4.943	2.694	3.975	3.521333333
24	4.946333333	2.708	3.574	3.699

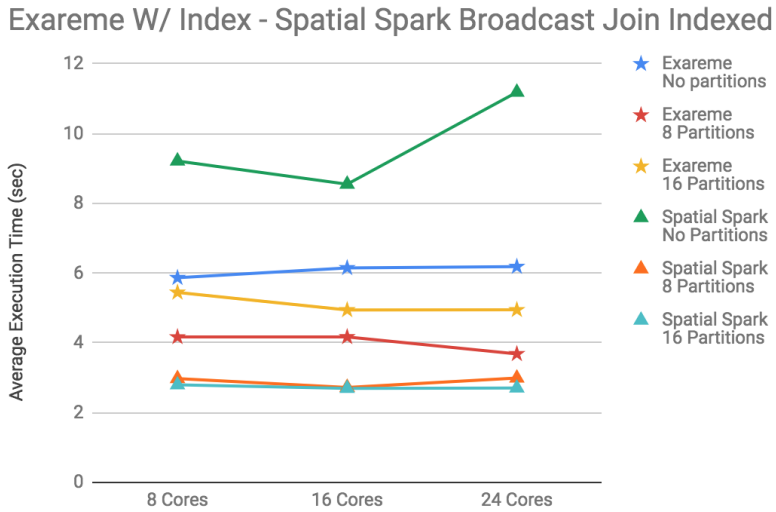

Figure 29: Polygon Contains Polygon, Comparing Exareme to Spatial Spark

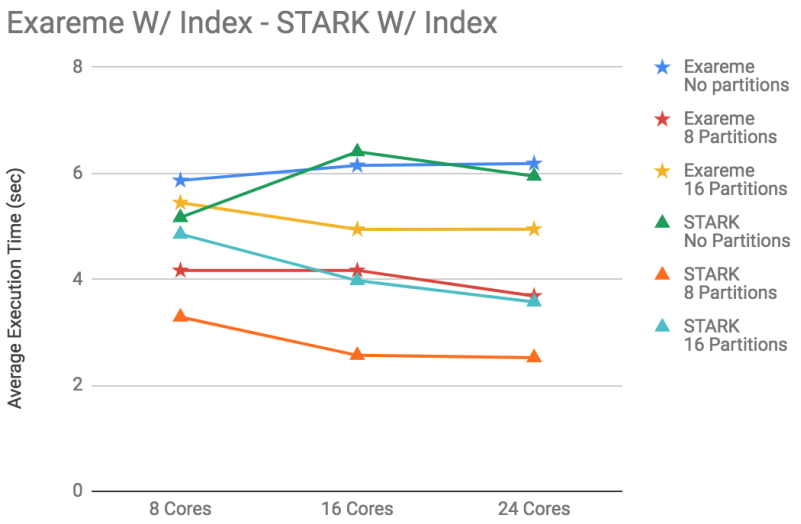

Figure 30: Polygon Contains Polygon, Comparing Exareme to STARK

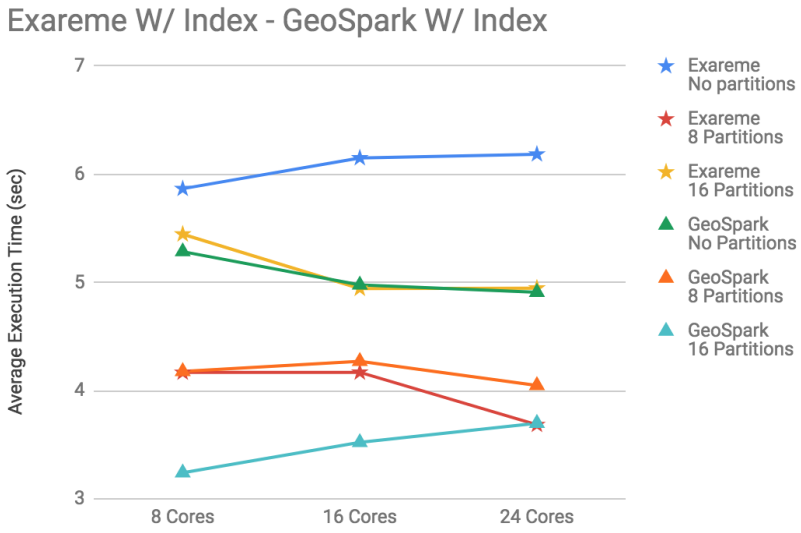

Figure 31: Polygon Contains Polygon, Comparing Exareme to GeoSpark

Table 15: Comparing systems without indexers, no partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	GeoSpark w/o Indexer
8	26.1217	5.837	16.997
16	28.2593	6.904	16.689
24	28.4877	7.116	16.71233333

Table 16: Comparing systems without indexers, 8 partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	GeoSpark w/o Indexer
8	13.247	3.61	4.890666667
16	11.2867	2.431	5.046333333
24	11.4163	2.573	4.318333333

Table 17: Comparing systems without indexers, 16 partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	GeoSpark w/o Indexer
8	8.014	4.798	10.332
16	7.53833	4.256	3.363333333
24	7.60033	3.671	3.699666667

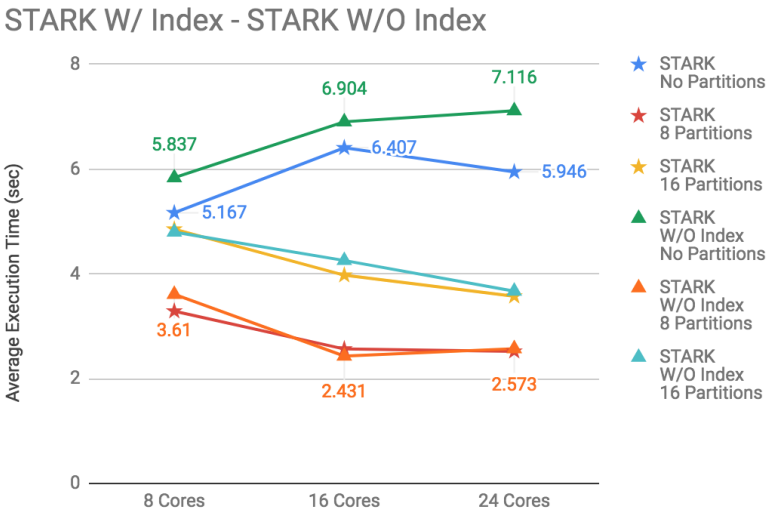

Figure 32: Polygon Contains Polygon, Comparing STARK W/ and W/O indexers

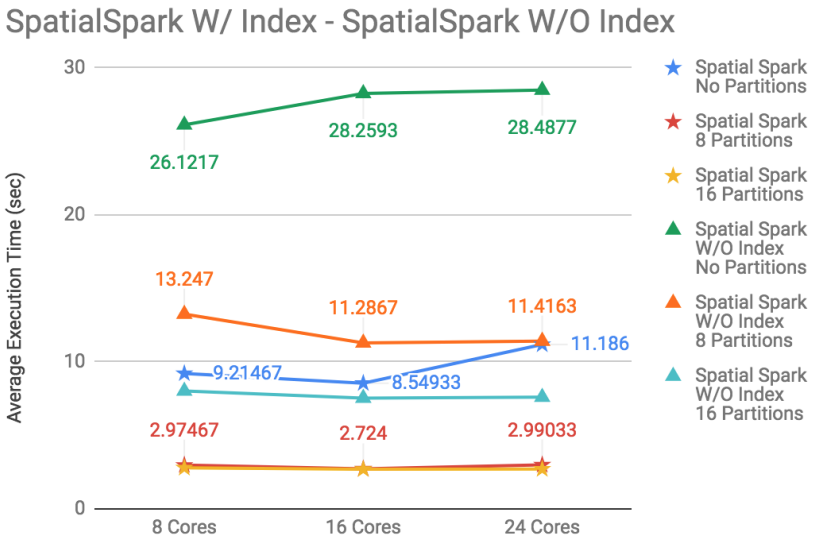

Figure 33: Polygon Contains Polygon, Comparing Spatial Spark W/ and W/O indexers

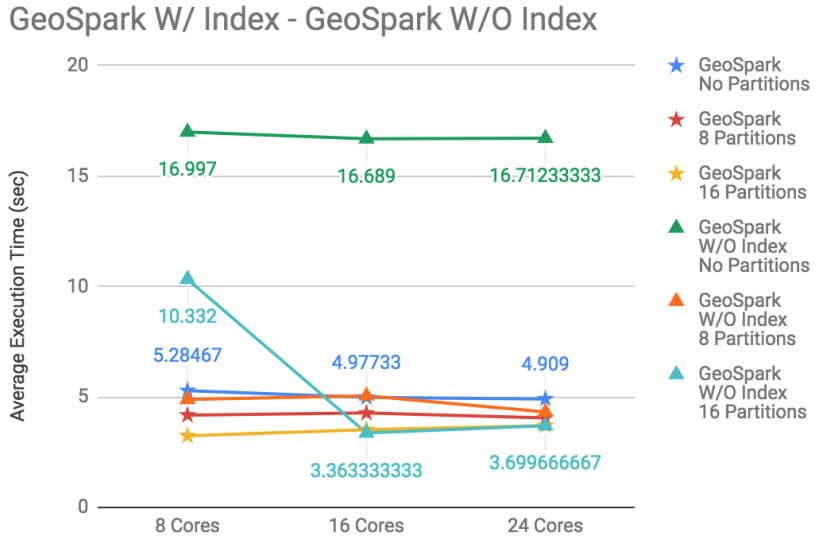

Figure 34: Polygon Contains Polygon, Comparing GeoSpark W/ and W/O indexers

4.2.2.3. Polygon Disjoint Polygon

Table 18: Comparing Exareme results with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	5.030666667	4.08	4.880333333
16	4.902	3.694666667	4.353333333
24	4.997333333	3.624	4.300333333

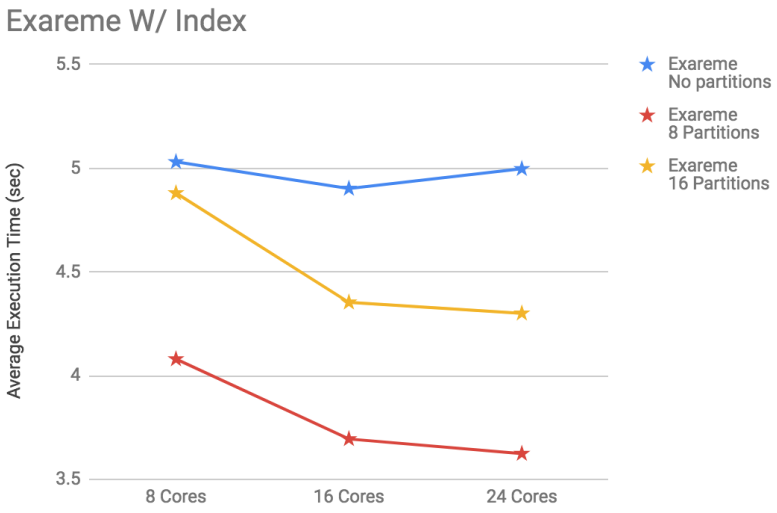

Figure 35: Polygon Disjoint Polygon, Exareme W/ indexer

4.2.2.4. Polygon Equals Polygon

Table 19: Comparing Exareme results with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	5.597666667	3.761	5.019666667
16	5.567333333	3.855333333	4.476
24	5.487666667	3.662666667	5.367

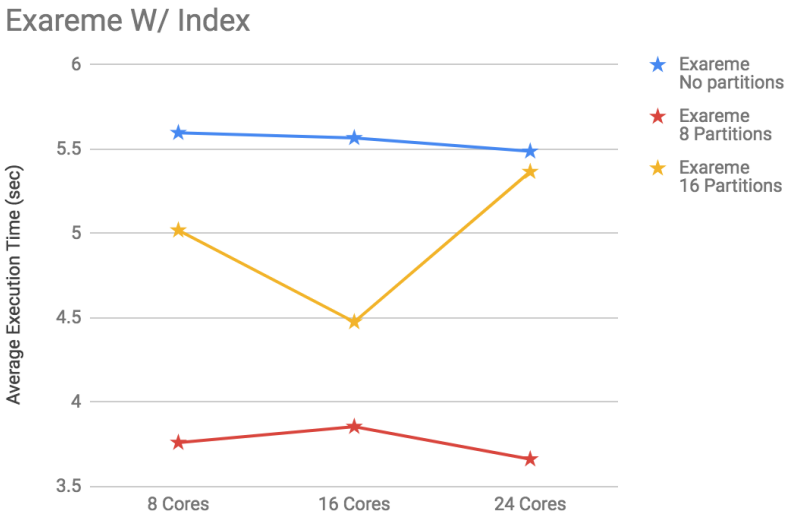

Figure 36: Polygon Equals Polygon, Exareme W/ indexer

4.2.2.5. Polygon Overlaps Polygon

Table 20: Comparing systems with indexers, no partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	6.415666667	8.68733
16	6.572666667	9.307
24	6.665666667	5.79767

Table 21: Comparing systems with indexers, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	4.018666667	2.95967
16	4.162333333	2.785
24	3.903333333	3.03667

Table 22: Comparing systems with indexers, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	5.244666667	2.92733
16	4.356666667	3.08467
24	4.801333333	3.069

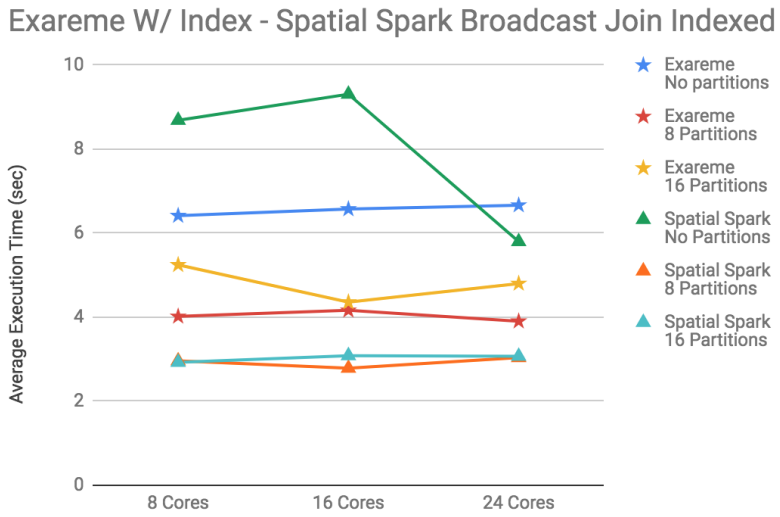

Figure 37: Polygon Overlaps Polygon, Exareme W/ indexer and Spatial Spark Broadcast Spatial Join W/ indexer

Table 23: Comparing Spatial Spark without indexers

Cores	Spatial-Spark Partitioned Spatial Join No Partitions	Spatial-Spark Partitioned Spatial Join 8 Partitions	Spatial-Spark Partitioned Spatial Join 16 Partitions
8	27.3463	12.361	8.30867
16	25.8323	12.372	8.25933
24	26.782	11.989	7.34933

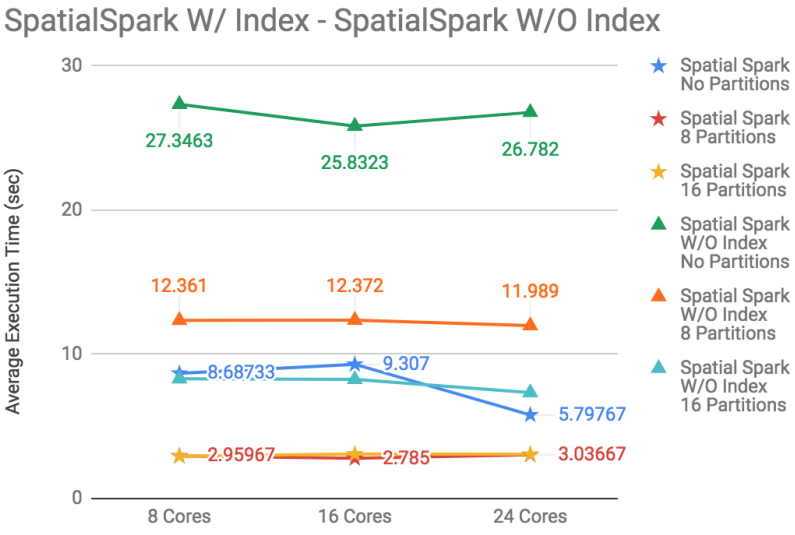

Figure 38: Polygon Overlaps Polygon, Spatial Spark W/ and W/O indexer

4.2.2.6. Polygon Touches Polygon

Table 24: Comparing Exareme with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	6.509	3.968	5.091
16	6.339333333	4.077	4.456
24	6.616666667	3.876666667	4.437333333

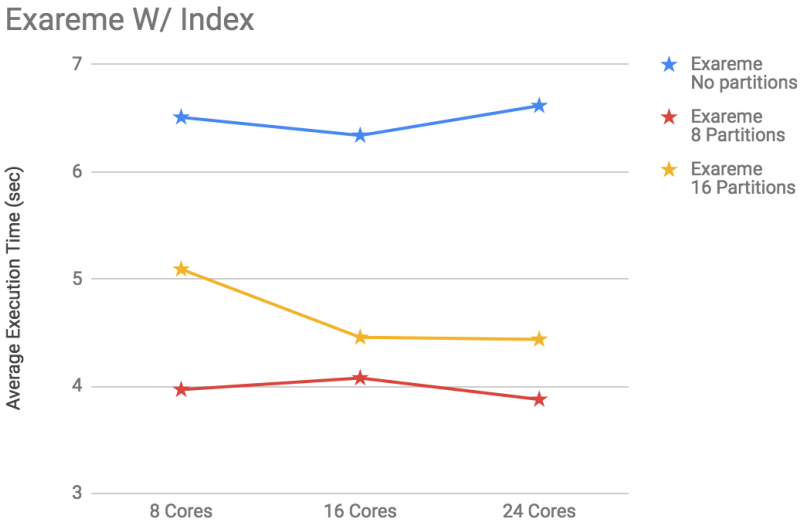

Figure 39: Polygon Touches Polygon, Exareme W/ indexer

4.2.2.7. Polygon Within Polygon

Table 25: Comparing systems with indexers, no partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	6.024333333	9.16567
16	5.830666667	8.032
24	6.105333333	9.29533

Table 26: Comparing systems with indexers, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	3.778	2.80133
16	4.215333333	2.80567
24	3.747	3.008

Table 27: Comparing systems with indexers, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	5.364333333	2.937
16	4.372	2.64567
24	4.266333333	2.80067

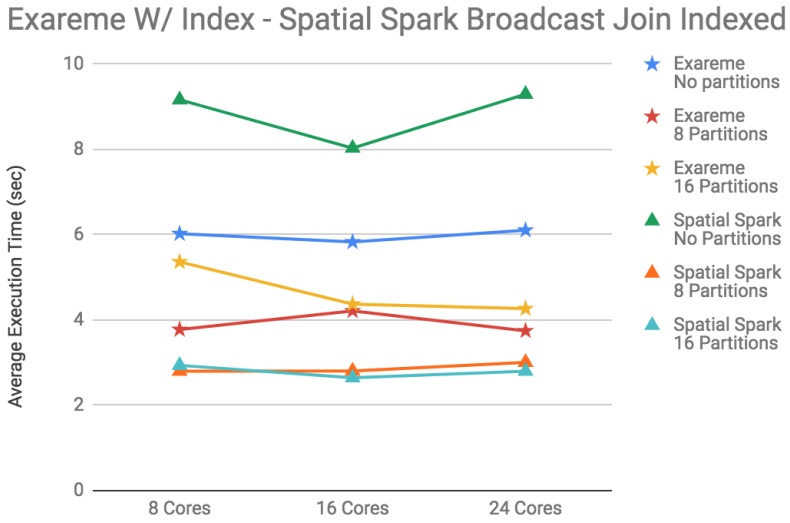

Figure 40: Polygon Within Polygon, Exareme W/ indexer and Spatial Spark Broadcast Spatial Join W/ indexer

Table 28: Comparing Spatial Spark W/ and W/O indexer

Cores	Spatial-Spark Partitioned Spatial Join No Partitions	Spatial-Spark Partitioned Spatial Join 8 Partitions	Spatial-Spark Partitioned Spatial Join 16 Partitions
8	24.1467	12.312	8.354
16	26.4613	11.939	7.44767
24	27.8107	11.2147	6.48867

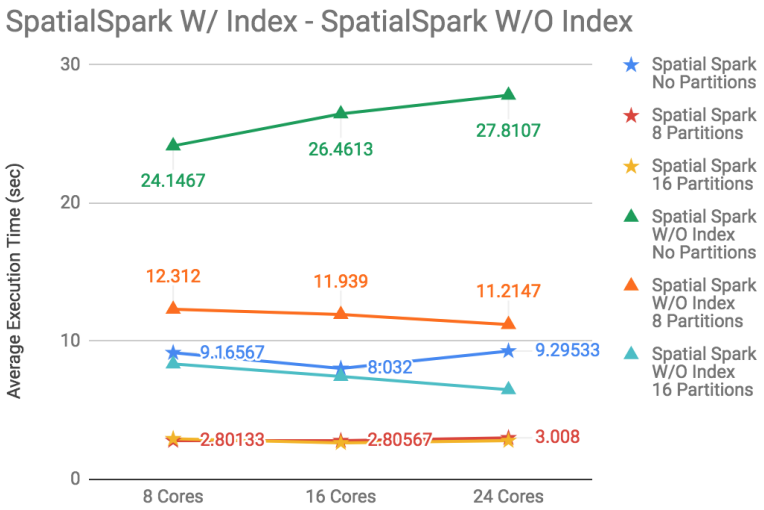

Figure 41: Polygon Within Polygon, Spatial Spark W/ and W/O indexer

4.2.2.8. Line Crosses Line

Table 29: Comparing Exareme with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8		669.1366667	
16	990.6183333		
24	463.6186667		510.046

Figure 42: Line Crosses Line, Exareme W/ indexer

4.2.2.9. Line Crosses Polygon

Table 30: Comparing Exareme with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	44.521	13.674	14.72133333
16	43.40633333	13.13933333	14.67
24	43.719	13.494	14.65033333

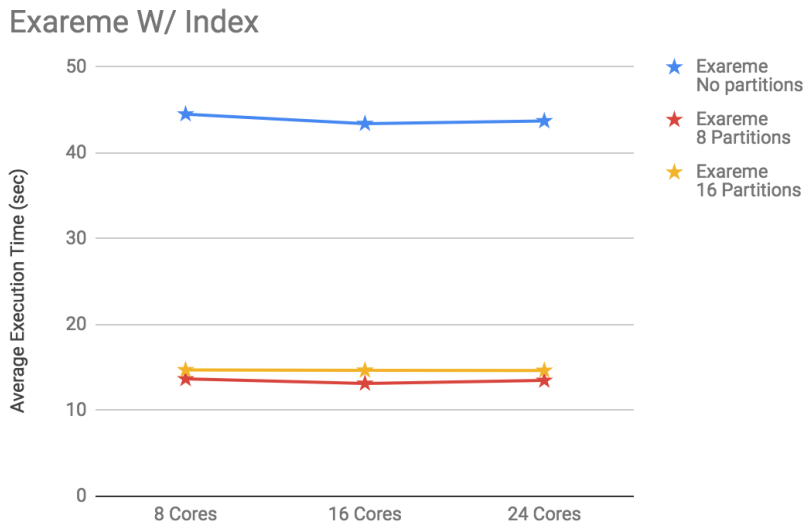

Figure 43: Line Crosses Polygon, Exareme W/ indexer

4.2.2.10. Line Intersects Line

Table 31: Spatial Spark with indexers

Cores	Spatial Spark Broadcast Spatial Join No partitions	Spatial Spark Broadcast Spatial Join 256 Partitions	Spatial Spark Broadcast Spatial Join 512 Partitions
8		1748.71	1533.694
16			
24			

Table 32: STARK with indexer

Cores	STARK LiveIndex No partitions	STARK LiveIndex 8 Partitions	STARK LiveIndex 16 Partitions
8	16.867	39.791	56.176
16	22.055	27.246	41.608
24	23.792	28.597	41.942

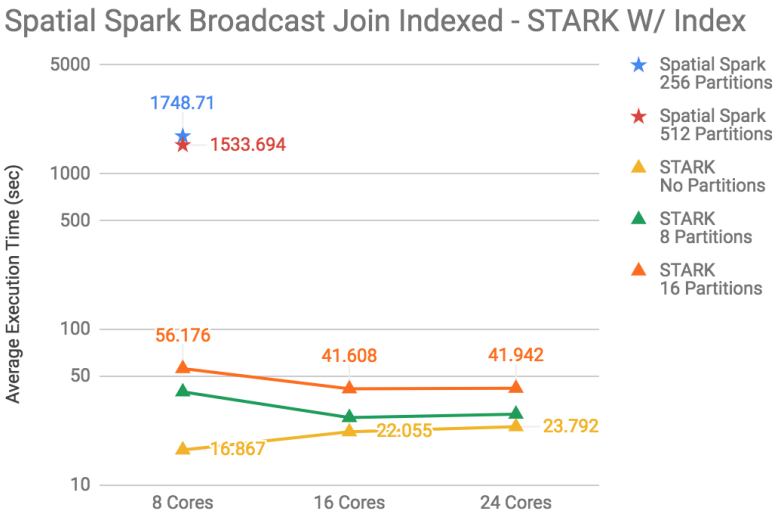

Figure 44: Line Intersects Line, Comparing systems W/ indexer

Table 33: Comparing STARK W/O indexer

Cores	STARK No Partitions	STARK 8 Partitions	STARK 16 Partitions
8	15.646	3876.38	455.907
16	14.78	4640.05	492.798
24	19.431	4193.12	546.562

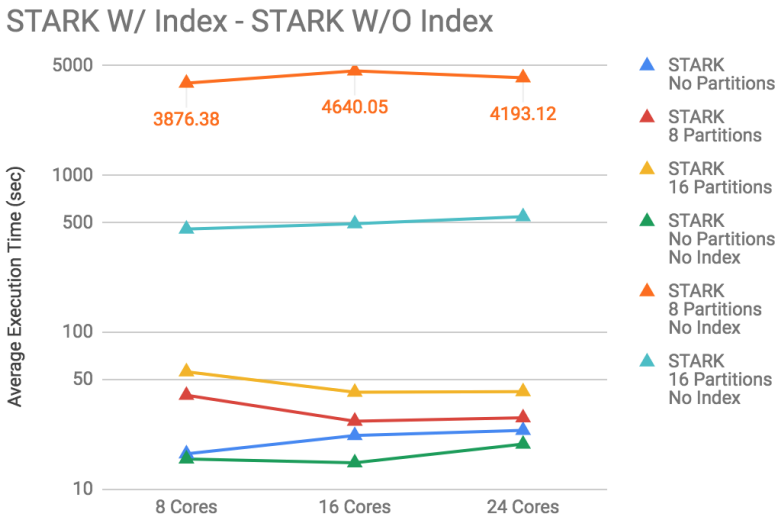

Figure 45: Line Intersects Line, STARK W/ and W/O indexer

4.2.2.11. Line Intersects Polygon

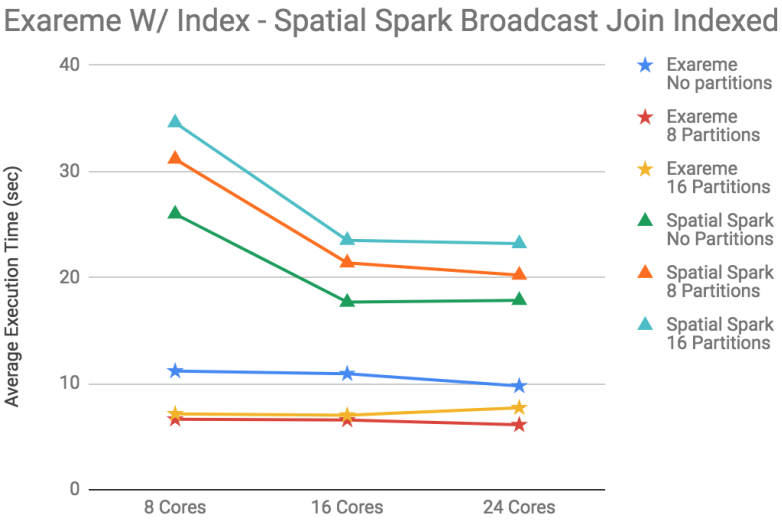

Figure 46: Line Intersects Polygon, Comparing Exareme W/ indexer with Spatial Spark

Table 34: Comparing systems with indexers, no partitions

Cores	Exareme SpatialIndex	STARK LiveIndex
8	11.18166667	63.44
16	10.932	85.7193
24	9.779333333	79.882

Table 35: Comparing systems with indexers, 8 partitions

Cores	Exareme SpatialIndex	STARK LiveIndex
8	6.643666667	172.071
16	6.570666667	183.619
24	6.114666667	196.867

Table 36: Comparing systems with indexers, 16 partitions

Cores	Exareme SpatialIndex	STARK LiveIndex
8	7.152333333	132.378
16	7.015333333	108.217
24	7.722333333	129.245

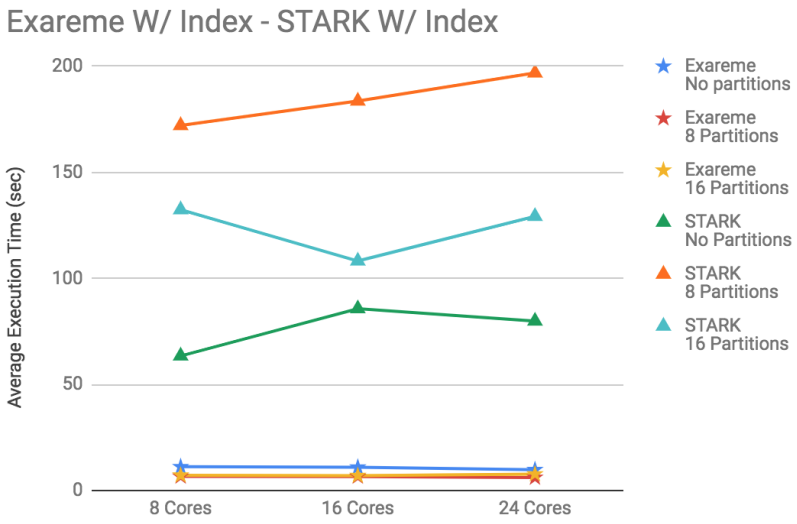

Figure 47: Line Intersects Polygon, Comparing Exareme W/ indexer with STARK

Table 37: Spatial Spark with indexers

Cores	Spatial Spark Broadcast Spatial Join No Partitions	Spatial Spark Broadcast Spatial Join 256 Partitions	Spatial Spark Broadcast Spatial Join 512 partitions
8	25.992	31.1643	34.5913
16	17.676	21.3833	23.5087
24	17.8473	20.2287	23.1947

Table 38: Spatial Spark without indexers

Cores	Spatial Spark Partitioned Spatial Join 64 partitions	Spatial Spark Partitioned Spatial Join 256 partitions	Spatial Spark Partitioned Spatial Join 512 partitions
8		176.241	190.424
16		165.314	138.988
24		173.717	127.113

Table 39: STARK without indexers

Cores	STARK No Partitions	STARK 8 Partitions	STARK 16 Partitions
8	672.017	269.777	285.78
16	647.499	209.012	210.202
24	509.661	158.358	191.217

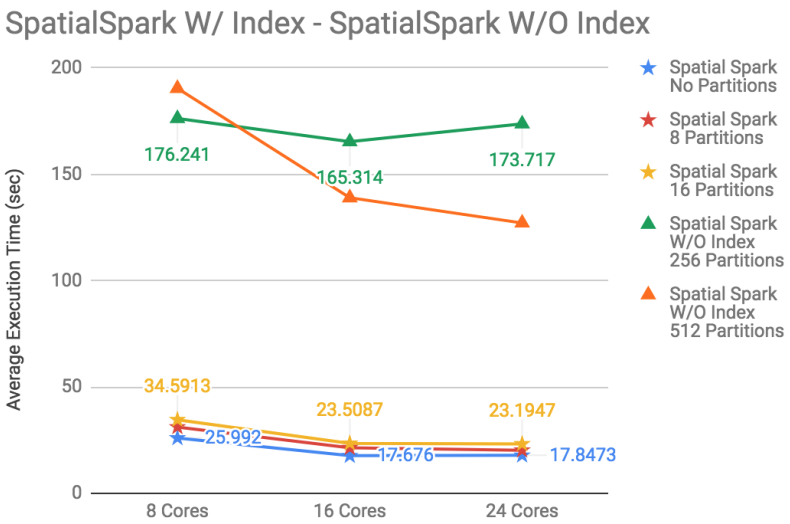

Figure 48: Line Intersects Polygon, Comparing Spatial Spark W/ and W/O indexer

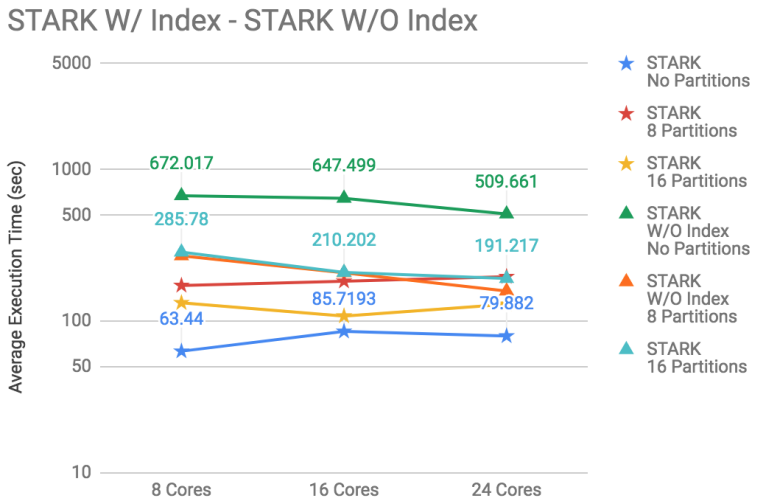

Figure 49: Line Intersects Polygon, Comparing STARK W/ and W/O indexer

4.2.2.12. Line Overlaps Polygon

Table 40: Comparing systems with indexers, no partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	GeoSpark Indexed
8	43.72133333	28.9673	153.0473333
16	43.31733333	21.186	
24	43.713	22.5777	

Table 41: Comparing systems with indexers, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	GeoSpark Indexed
8	13.437	28.0183	
16	13.09566667	21.9503	143.327
24	14.256	19.3307	127.7576667

Table 42: Comparing systems with indexers, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	GeoSpark Indexed
8	14.64333333	28.5523	161.2096667
16	14.97733333	20.3847	105.4203333
24	14.29566667	24.9103	

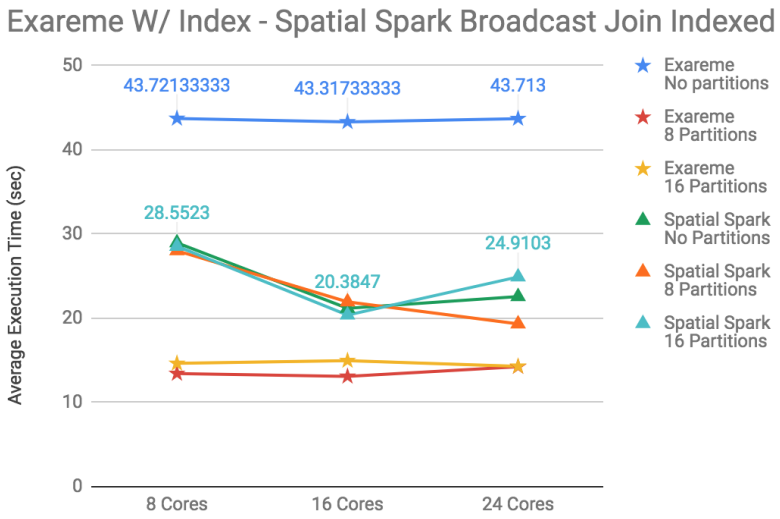

Figure 50: Line Overlaps Polygon, Comparing Exareme W/ indexer with Spatial Spark

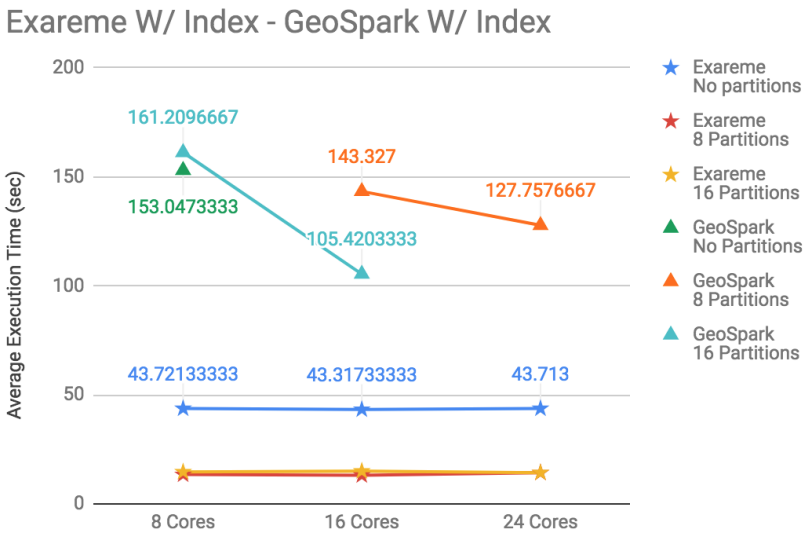

Figure 51: Line Overlaps Polygon, Comparing Exareme W/ indexer with GeoSpark

Table 43: Spatial Spark without indexers

Cores	Spatial Spark Partitioned Spatial Join 64 partitions	Spatial Spark Partitioned Spatial Join 256 partitions	Spatial Spark Partitioned Spatial Join 512 partitions
8	222.878	171.346	180.521
16		135.5	130.491
24		149.45	119.093

Table 44: GeoSpark without indexer

Cores	GeoSpark No Partitions	GeoSpark 8 Partitions	GeoSpark 16 Partitions
8	1144.688333		365.7136667
16	1049.238	1237.103667	356.5803333
24	1078.862333	1469.978333	402.1166667

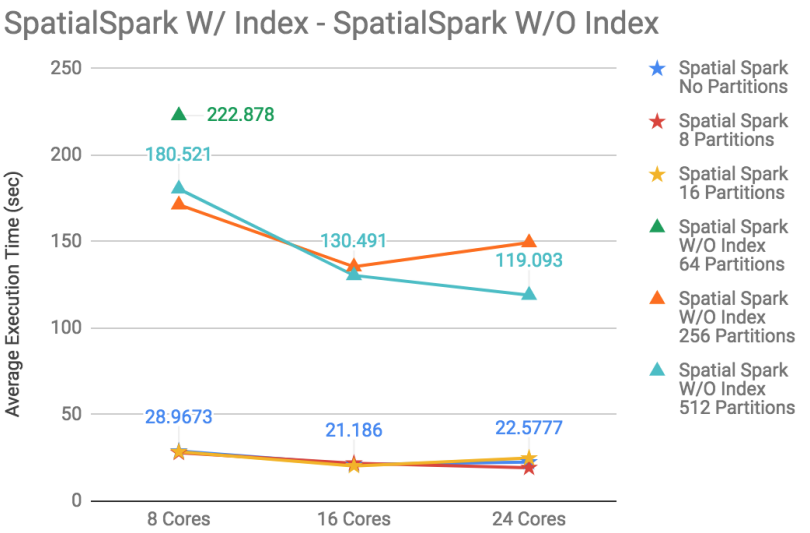

Figure 52: Line Overlaps Polygon, Comparing Spatial Spark W/ and W/O indexer

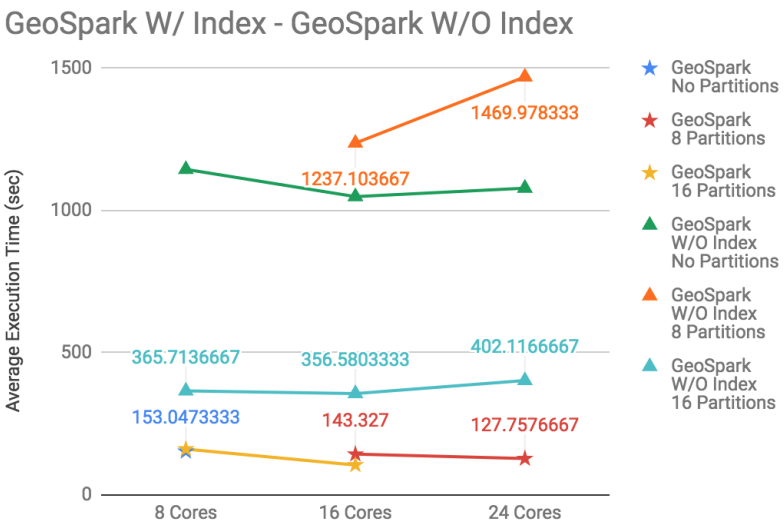

Figure 53: Line Overlaps Polygon, Comparing GeoSpark W/ and W/O indexer

4.2.2.13. Line Touches Polygon

Table 45: Comparing Exareme with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	45.757	14.72033333	15.956
16	45.62733333	14.705	16.04433333
24	44.50833333	14.32133333	15.783

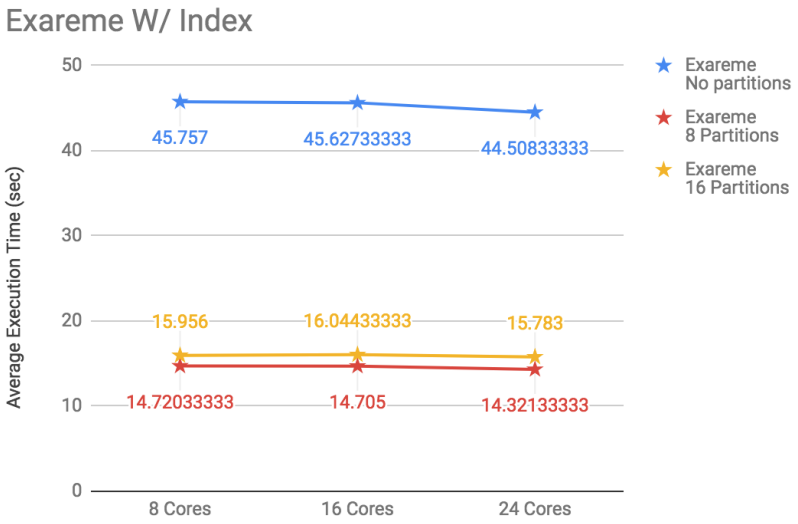

Figure 54: Line Touches Polygon, Exareme W/ indexer

4.2.2.14. Line Within Polygon

Table 46: Exareme with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	18.52633333	6.662333333	8.343333333
16	19.06266667	7.655666667	8.143
24	18.192	6.652666667	7.581666667

Table 47: Magellan with indexer

Cores	Magellan Indexed No Partitions
8	7.309
16	8.958
24	7.885

Table 48: Spatial Spark with indexer

Cores	Spatial Spark Broadcast Spatial Join No Partitions	Spatial Spark Broadcast Spatial Join 64 Partitions	Spatial Spark Broadcast Spatial Join 256 Partitions
8		25.5593	29.6363
16		17.6893	21.223
24		18.734	21.156

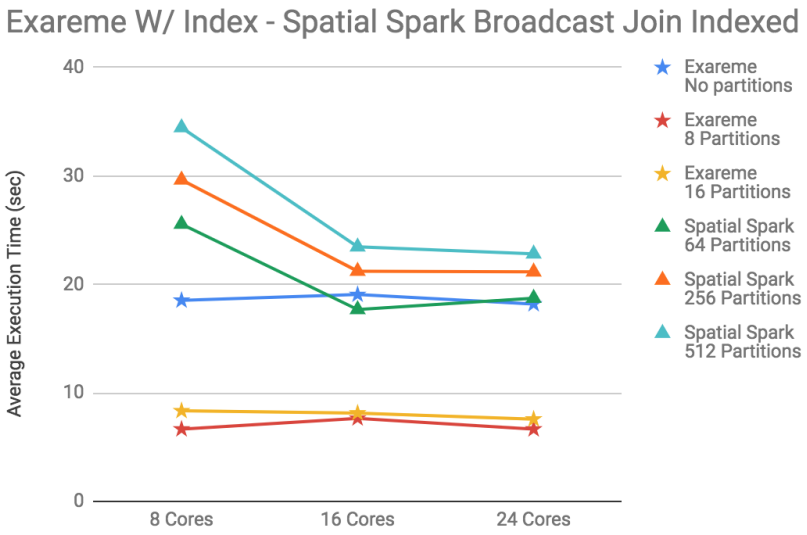

Figure 55: Line Within Polygon, Comparing Exareme W/ indexer with Spatial Spark

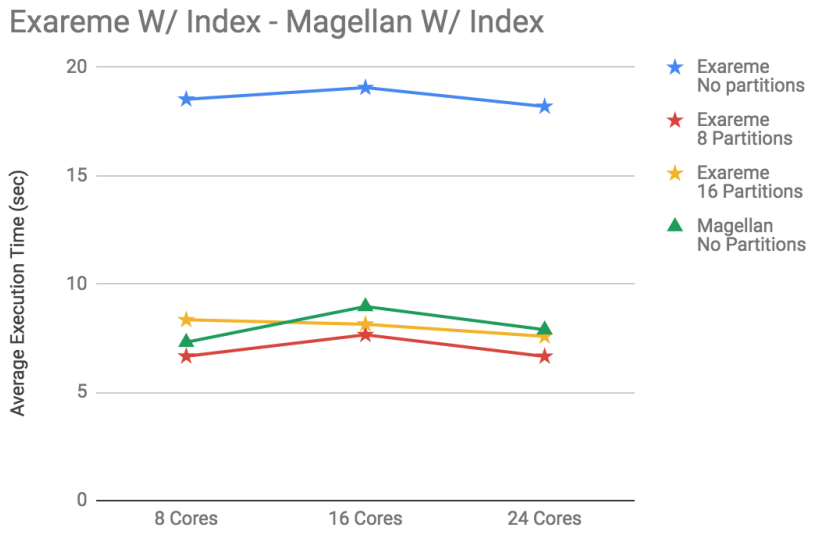

Figure 56: Line Within Polygon, Comparing Exareme W/ indexer with Magellan

Table 49: Magellan without indexer

Cores	Magellan
8	18.748
16	19.655
24	18.221

Table 50: Spatial Spark without indexer

Cores	Spatial Spark Partitioned Spatial Join No Partitions	Spatial Spark Partitioned Spatial Join 256 Partitions	Spatial Spark Partitioned Spatial Join 512 Partitions
8		172.867	187.909
16		135.273	134.413
24		149.172	122.559

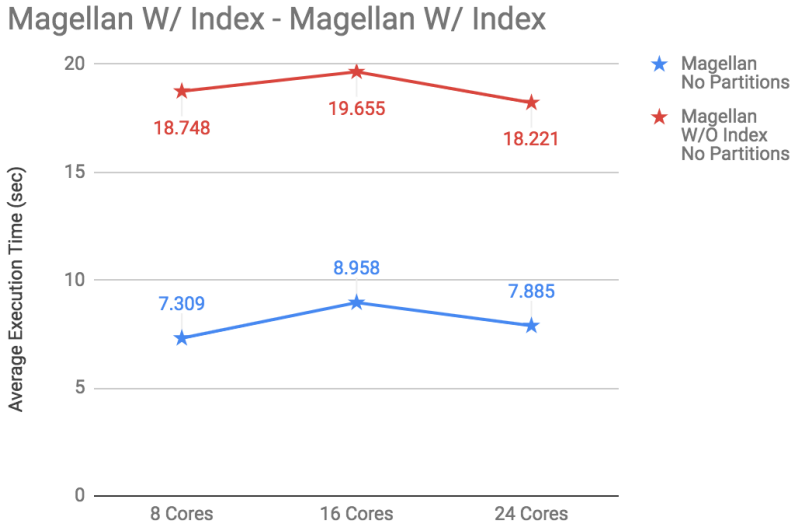

Figure 57: Line Within Polygon, Comparing Magellan W/ and W/O indexer

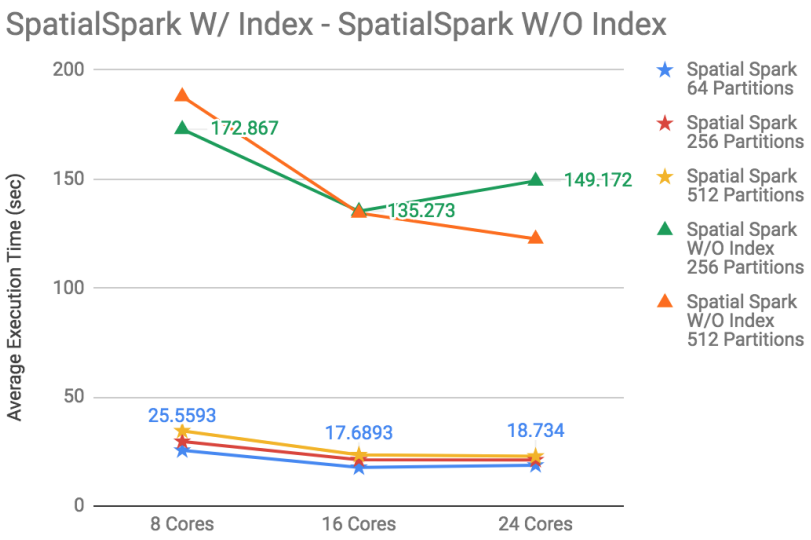

Figure 58: Line Within Polygon, Comparing Spatial Spark W/ and W/O indexer

4.2.2.15. Point Equals Point

Table 51: Exareme with indexer

Cores	Exareme SpatialIndex No Partitions	Exareme SpatialIndex 8 Partitions	Exareme SpatialIndex 16 Partitions
8	3.610333333	3.184	4.785
16	3.629	3.515666667	4.320333333
24	3.559666667	2.982333333	3.802

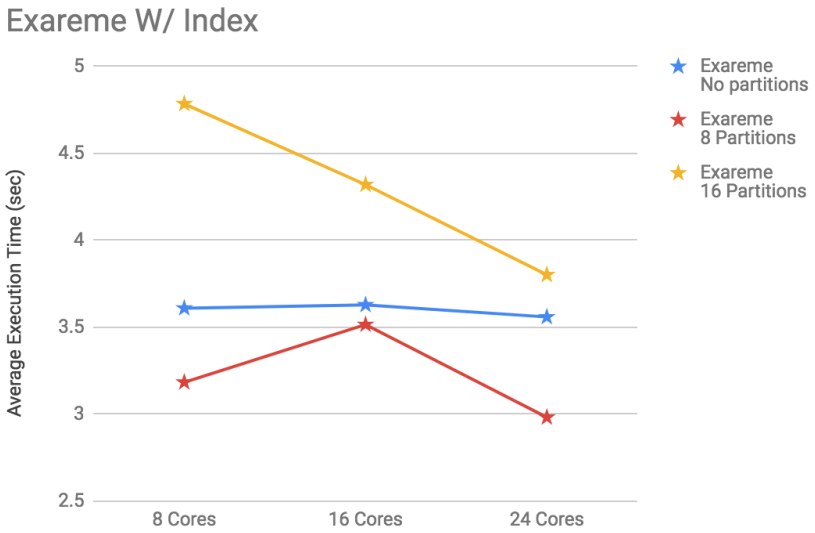

Figure 59: Point Equals Point, Exareme W/ indexer

4.2.2.16. Point Intersects Line

Table 52: Compare systems with indexer, No partitions

Cores	Exareme SpatialIndex	STARK LiveIndex
8	5.230333333	58.169
16	5.324666667	78.7197
24	5.582	84.255

Table 53: Compare systems with indexer, 8 partitions

Cores	Exareme SpatialIndex	STARK LiveIndex
8	3.76	164.792
16	3.674333333	181.174
24	3.544666667	206.851

Table 54: Compare systems with indexer, 16 partitions

Cores	Exareme SpatialIndex	STARK LiveIndex
8	5.134	130.847
16	4.623	101.258
24	4.660333333	106.007

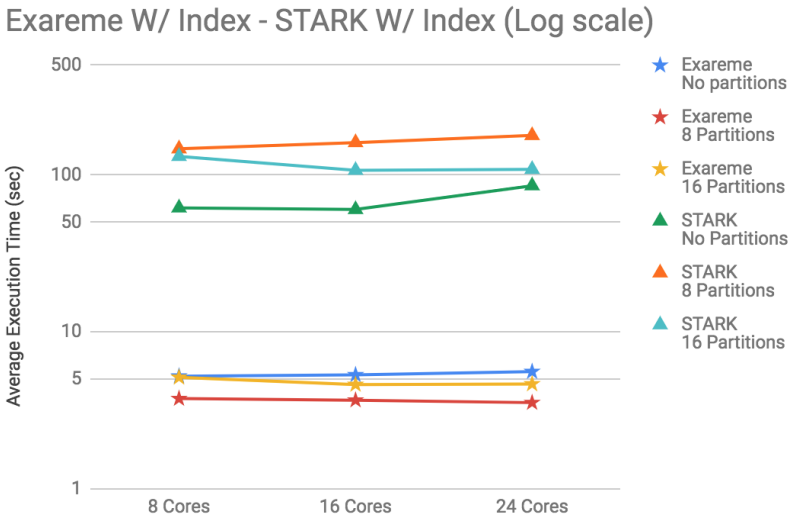

Figure 60: Point Intersects Line, Systems W/ indexer

Table 55: STARK w/o indexers

Cores	STARK w/o Indexer No Partitions	STARK w/o Indexer 8 Partitions	STARK w/o Indexer 16 Partitions
8	1568.74	287.395	251.779
16	1082.6	201.949	168.79
24	1320.85	197.938	152.729

Table 56: Spatial Spark w/o indexers

Cores	Spatial Spark Partitioned Spatial Join No Partitions	Spatial Spark Partitioned Spatial Join 64 Partitions	Spatial Spark Partitioned Spatial Join 256 Partitions
8		201.138	178.307
16		226.744	124.014
24			143.49

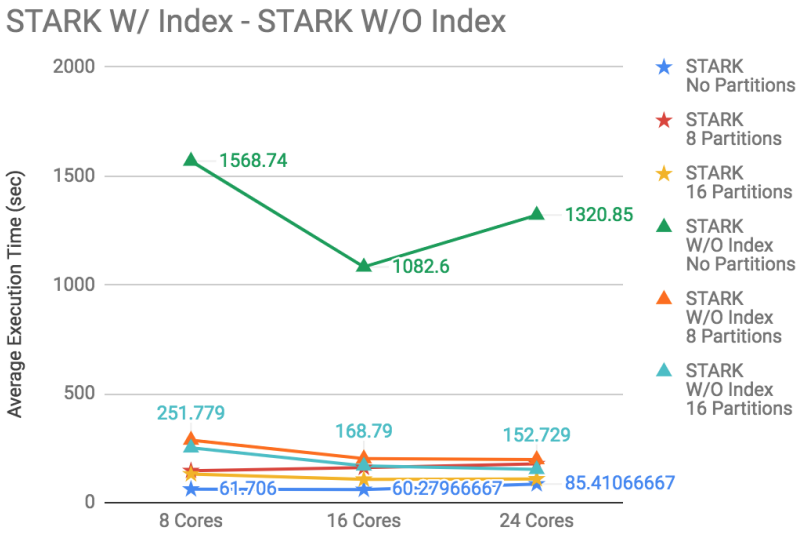

Figure 61: Point Intersects Line, STARK W/ and W/O indexer

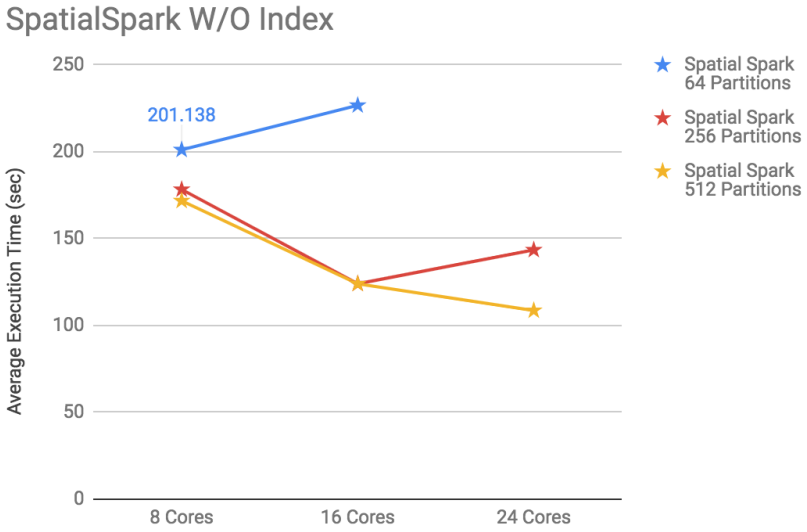

Figure 62: Point Intersects Line, Spatial Spark W/O indexer

4.2.2.17. Point Intersects Polygon

Table 57: Compare systems with indexer, No partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex	Magellan Indexed
8	2.966	6.00233	1.120666667	31.706
16	3.114666667	8.96567	3.878333333	27.765
24	2.715333333	7.30667	3.802666667	30.854

Table 58: Compare systems with indexer, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex
8	2.841	2.231	2.116666667
16	3.199333333	2.13033	1.625
24	3.163666667	2.39267	1.870666667

Table 59: Compare systems with indexer, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	STARK LiveIndex
8	4.565666667	2.17033	3.794666667
16	3.994333333	2.29233	3.029333333
24	3.870666667	2.07867	3.083333333

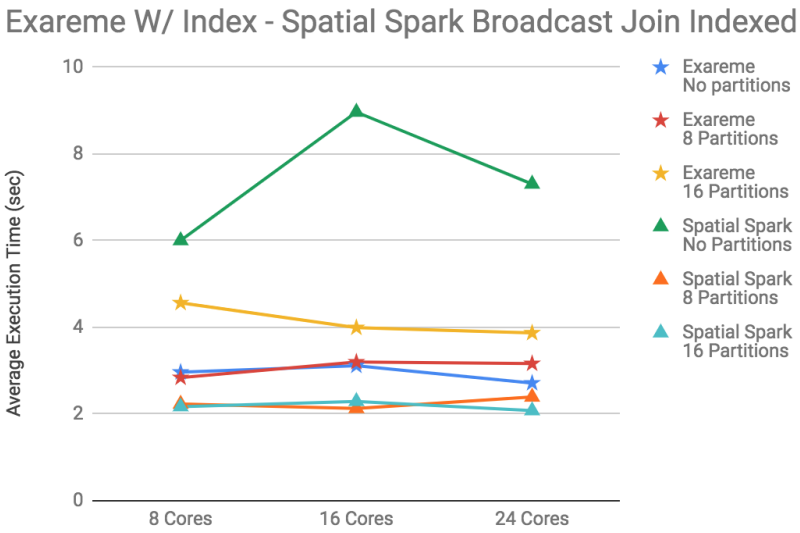

Figure 63: Point Intersects Polygon, Systems W/ indexer with Spatial Spark

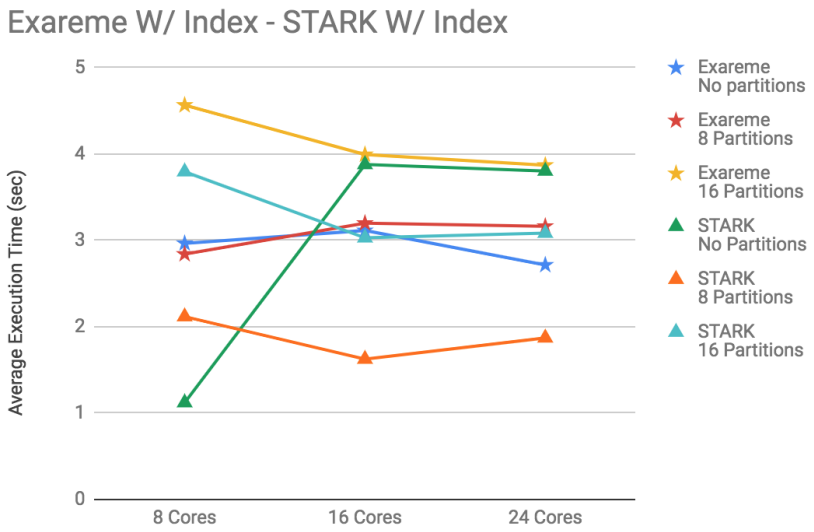

Figure 64: Point Intersects Polygon, Systems W/ indexer with STARK

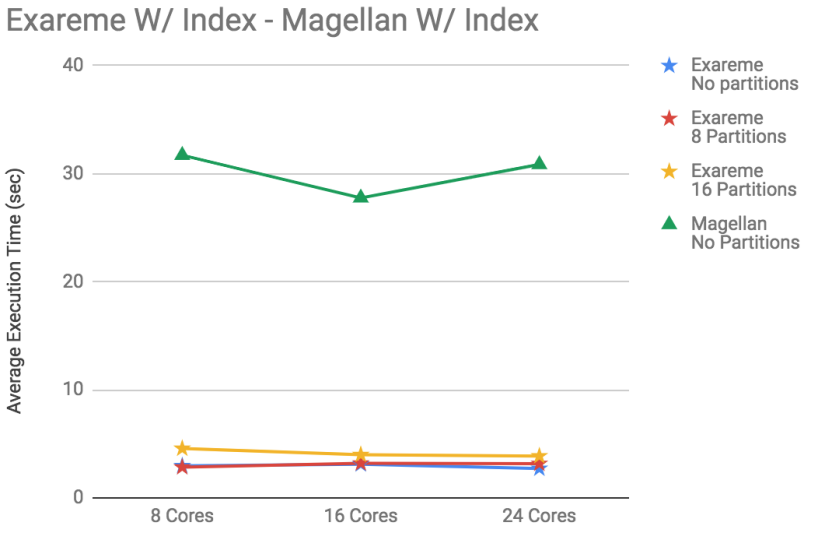

Figure 65: Point Intersects Polygon, Systems W/ indexer with Magellan

Table 60: Compare systems W/O indexer, No partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer	Magellan w/o Indexer
8	15.0203	6.54	31.662
16	14.3707	7.091	33.652
24	13.3583	6.883666667	26.611

Table 61: Compare systems W/O indexer, 8 partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer
8	8.358	2.985
16	7.176	1.996
24	7.521	2.432666667

Table 62: Compare systems W/O indexer, 16 partitions

Cores	Spatial-Spark Partitioned Spatial Join	STARK w/o Indexer
8	5.207	3.99
16	4.683	3.591333333
24	4.16533	3.290333333

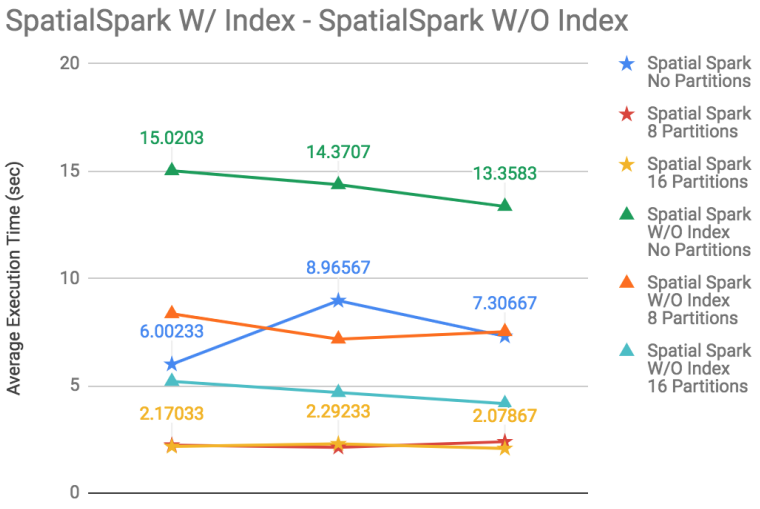

Figure 66: Point Intersects Polygon, Spatial Spark W/ and W/O indexers

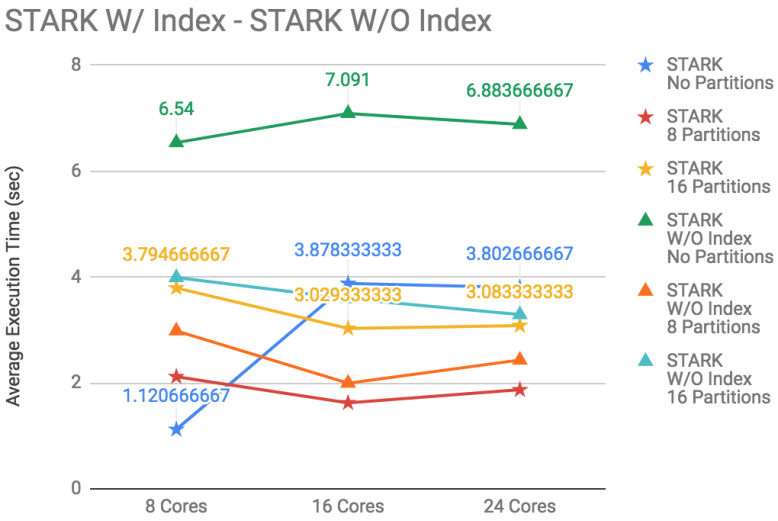

Figure 67: Point Intersects Polygon, STARK W/ and W/O indexers

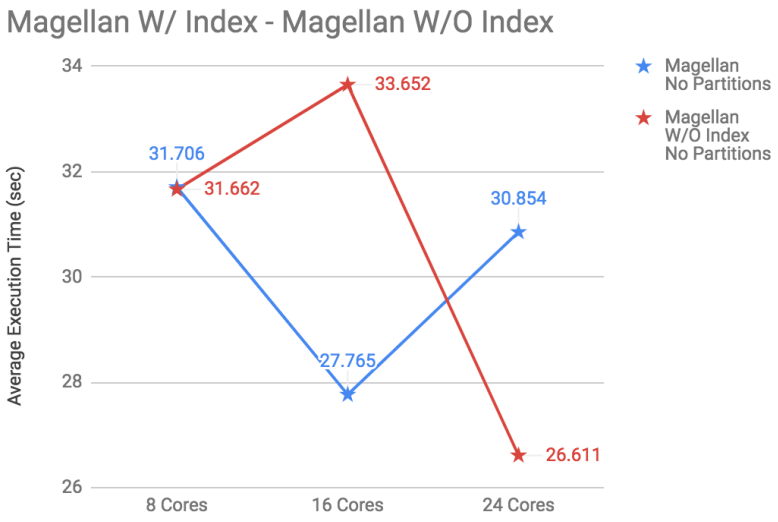

Figure 68: Point Intersects Polygon, Magellan W/ and W/O indexers

4.2.2.18. Point Within Polygon

Table 63: Compare systems with indexer, No partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join	Magellan Indexed
8	3.005333333	5.789	8.31
16	2.924	6.43	8.222
24	2.963	6.61433	8.796

Table 64: Compare systems with indexer, 8 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	3.129666667	2.096
16	2.873	2.119
24	3.194666667	2.174

Table 65: Compare systems with indexer, 16 partitions

Cores	Exareme SpatialIndex	Spatial Spark Broadcast Spatial Join
8	4.909333333	2.25
16	4.232333333	2.03067
24	3.884666667	2.082

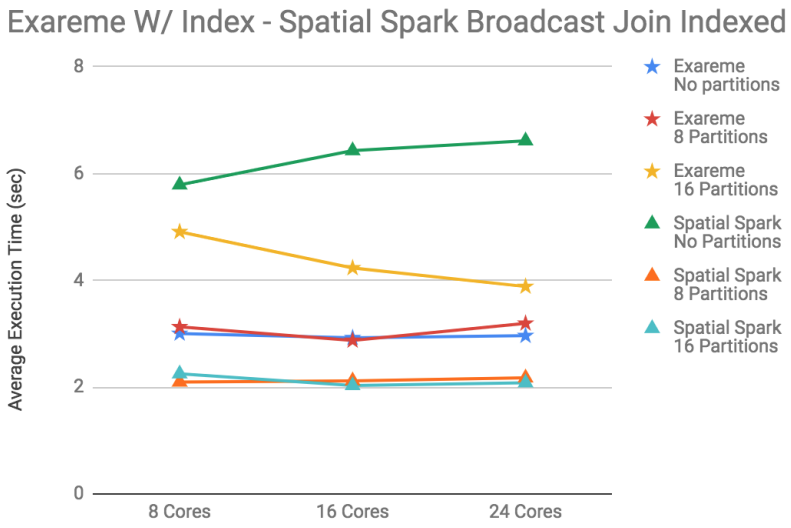

Figure 69: Point Within Polygon, Exareme W/ indexer with Spatial Spark

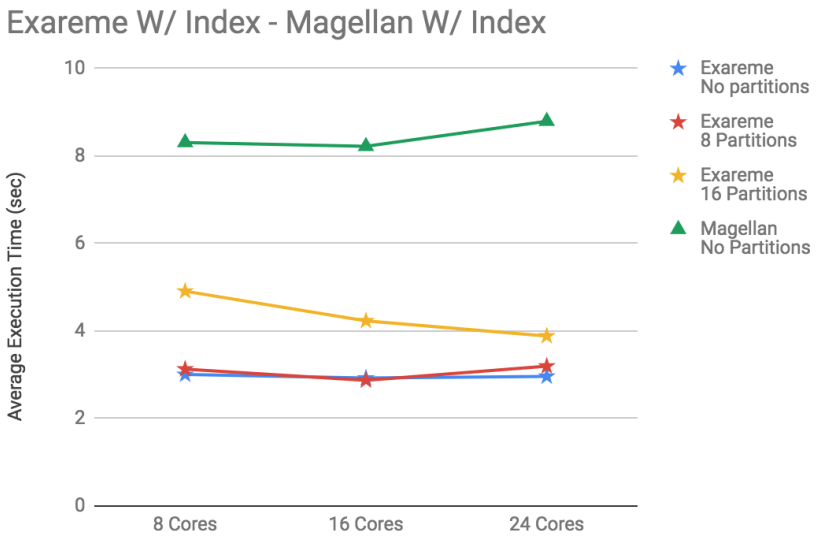

Figure 70: Point Within Polygon, Exareme W/ indexer with Magellan

Table 66: Magellan W/O indexer, No partitions

Cores	Magellan w/o Indexer
8	19.141
16	20.783
24	17.362

Table 67: Spatial Spark W/O indexer

Cores	Spatial-Spark Partitioned Spatial Join No Partitions	Spatial-Spark Partitioned Spatial Join 8 Partitions	Spatial-Spark Partitioned Spatial Join 16 Partitions
8	12.314	7.08133	5.31833
16	15.449	7.38333	6.01167
24	15.5243	7.486	4.48633

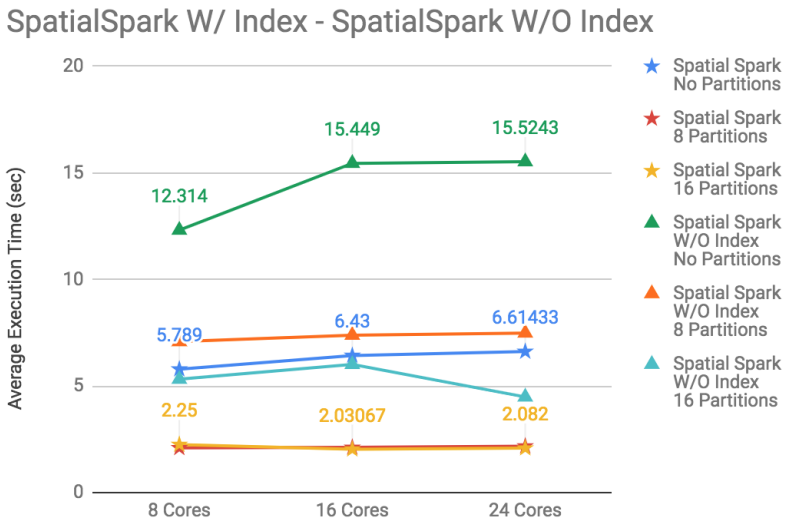

Figure 71: Point Within Polygon, Spatial Spark W/ and W/O indexers

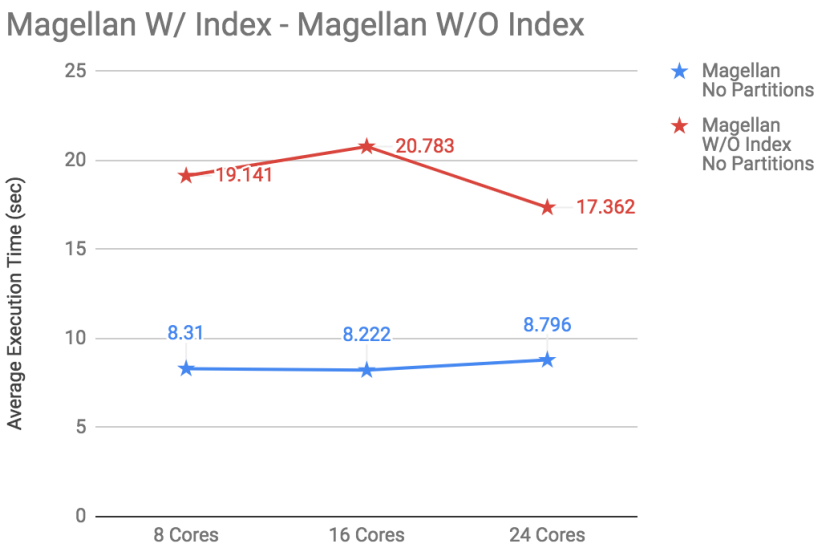

Figure 72: Point Within Polygon, Magellan W/ and W/O indexers

4.3. Challenges

For STARK, GeoSpark and Spatial-Spark, in the operations that are using Line objects (*edgeslm_merge* dataset) we changed the memory of Spark executors and that of the system. That was done because of the size of the dataset and the additional need in memory. Practically, we increased the amount of memory to use per executor process (`spark.executor.memory`) to 5 GB and decreased the memory to use for the driver process (`spark.driver.memory`) from 10GB to 4GB. This decrease was necessary for balancing the memory of the systems.

STARK's Grid partitioner currently supports 2 dimensional grids. Setting the number of partitions we practically set the number of cells in each side of the grid per dimension. So, a nominal of 8 partitions, corresponds to an 8 x 8 grid, rather than 8 distinguished partitions as Exareme does.

Spatial-Spark in Broadcast Spatial Join, couldn't fit *edges_merge* to memory as the right member of a spatial relationship. In order to run the tests, it needed to be partitioned in more than 8 or 16 partitions. We took advantage of STARK's fixed grid logic and we replicated that by giving 64 and 256 partitions correspondingly to the partitioners.

Spatial-Spark can perform worse on multiple computing nodes than on a single node, especially for experiments that are more data intensive. The low scalability may indicate that data communication overheads among distributed computing nodes might be a potential bottleneck.

In GeoSpark, in operations with Line objects index had to be persisted to disk because of memory issues that couldn't be resolved by just adjusting the memory properties for the Spark jobs.

In Exareme the partitioning was performed during the dataset loading. Dataset *edges_merge* took the most time due to its size, exceeding more than 2 hours. The experiment Line Crosses Line gave partial results and experiment Line intersects Line gave no results exceeding the timeout limit.

5. CONCLUSIONS

In this paper we conducted a detailed functional and performance evaluation to some of the most modern and complete state-of-the-art geospatial distributed systems. It is the first time that a well-established benchmark like Jackpine is used to evaluate the performance of distributed systems with spatial support. The outcome of the benchmarks shows that each system supports different spatial operations that can be used in SQL queries. STARK achieved great performance but it was supporting only some operations. On the other hand, Exareme supports all the operations and shows stable performance in query execution times but in most cases a bit less than the rest of the systems. Spatial Spark achieves great query execution times when datasets fit into the memory, but performance decreases when data is too large to fit in memory. Finally, GeoSpark and Magellan are among the weakest of the systems due to the poor support of operations, while Magellan doesn't even support spatial partitioning.

ABBREVIATIONS - ACRONYMS

GPS	Global Positioning System
SOS	Sensor Observation Services
AIS	Automatic Identification System
GIS	Geographic Information Systems
OGC	Open Geospatial Consortium
RDBMS	Relational DataBase Management System
DSL	Domain Specific Language
JDBC	Java DataBase Connector
APSW	Another Python SQL Wrapper
RDD	Resilient Distributed Dataset
SRDD	Spatial Resilient Distributed Dataset
KNN	K-Nearest Neighbours
OSM-XML	Open Street Map XML
WKT	Well Known Text
DE-9IM	Dimensionally Extended Nine-Intersection Model
DAG	Data Acyclic Graph
UDF	User Defined Function
ExaQL	Exareme Query Language
ExaDFL	Exareme DataFLows
VM	Virtual Machine
HDFS	Hadoop Distributed File System
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
OWL	Web Ontology Language

REFERENCES

[1] J. Bao, C. Sengstock, M. E. Ali, Y. Huang, M. Gertz, M. Renz, and J. Sankaranarayanan, editors. *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA, November 3-6, 2015*. ACM, 2015.

[2] M. A. Beyer and D. Laney. The importance of 'big data': A definition". Gartner, 2012.

[3] Y. Chronis, Y. Foufoulas, V. Nikolopoulos, A. Papadopoulos, L. Stamatogiannakis, C. Svingos, and Y. E. Ioannidis. A relational approach to complex dataflows. In *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016.*, 2016.

[4] A. Eldawy and M. F. Mokbel. The era of big spatial data: A survey. *Foundations and Trends in Databases*, 6(3-4):163–273, 2016.

[5] G. Garbis, K. Kyzirakos, and M. Koubarakis. Geographica: A Benchmark for Geospatial RDF Stores. ISWC 2013.

[6] H. Garcia-Molina, J. D. Ullman, and J. Widom. Database system implementation. Prentice Hall, 1999.

[7] S. Hagedorn, P. Götze, and K. Sattler. Big spatial data processing frameworks: Feature and performance evaluation. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, pages 490–493, 2017.

[8] S. Hagedorn and T. Räth. Efficient spatio-temporal event processing with STARK. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, pages 570–573, 2017.

[9] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan. MR-DBSCAN: a scalable mapreduce-based DBSCAN algorithm for heavily skewed data. *Frontiers Comput. Sci.*, 8(1):83–99, 2014.

[10] S. T. Leutenegger, J. M. Edgington, and M. A. López. STR: A simple and efficient algorithm for r-tree packing. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997, Birmingham, UK*, pages 497–506, 1997.

[11] S. Ray, B. Simion, and A. D. Brown. Jackpine: A benchmark to evaluate spatial database performance. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1139–1150, April 2011.

[12] G. Spiliopoulos, K. Chatzikokolakis, D. Zissis, E. Biliri, D. Papaspyros, G. Tsapelas, and S. Mouzakitis. Knowledge extraction from maritime spatiotemporal data: An evaluation of clustering algorithms on big data. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1682–1687, Dec 2017.

[13] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1071–1085, 2016.

[14] S. You, J. Zhang, and L. Gruenwald. Large-scale spatial join query processing in cloud. In *31st IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2015, Seoul, South Korea, April 13-17, 2015*, pages 34–41, 2015.