A Performance Study of Big Spatial Data Systems

Md Mahbub Alam, Suprio Ray, Virendra C. Bhavsar
University of New Brunswick, Fredericton, Canada
{malam5,sray,bhavsar}@unb.ca

ABSTRACT

With the accelerated growth in spatial data volume, being generated from a wide variety of sources, the need for efficient storage, retrieval, processing and analyzing of spatial data is ever more important. Hence, spatial data processing system has become an important field of research. In recent times a number of Big Spatial Data systems have been proposed by researchers around the world. These systems can be roughly categorized into Apache Hadoop-based and in-memory systems based on Apache Spark. The available features supported by these systems vary widely. However, there has not been any comprehensive evaluation study of these systems in terms of performance, scalability and functionality. To address this need, we propose a benchmark to evaluate Big Spatial Data systems.

Although, Spark is a very popular framework, its performance is limited by the overhead associated with distributed resource management and coordination. The Big Spatial Data systems that are based on Spark, are also constrained by these. We introduce SpatialIgnite, a Big Spatial Data system that we have developed based on Apache Ignite. We investigate the present status of the Big Spatial Data systems by conducting a comprehensive feature analysis and performance evaluation of a few representative systems with our benchmark. Our study shows that SpatialIgnite performs better than Hadoop and Spark based systems that we have evaluated.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**; *Parallel and distributed DBMSs*;

KEYWORDS

Big Spatial Data, Hadoop, Spark, In-Memory, Ignite, Benchmark, Performance Evaluation

ACM Reference format:
Md Mahbub Alam, Suprio Ray, Virendra C. Bhavsar . 2018. A Performance Study of Big Spatial Data Systems. In *Proceedings of 7th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, Seattle, WA, USA, November 6, 2018 (BigSpatial 2018),* 9 pages.
https://doi.org/10.1145/3282834.3282841

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
BigSpatial 2018, November 6, 2018, Seattle, WA, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-6041-8/18/11…$15.00
https://doi.org/10.1145/3282834.3282841

1 INTRODUCTION

The volume of spatial data generated and consumed is rising rapidly. The popularity of location-based services and applications like Google Maps, vehicle-navigation-systems, recommendation systems, and location-based social networks are contributing to this data growth. In addition, spatial data is being generated from sources as diverse as medical devices, satellites, space telescopes, climate simulation, and oceanography to name a few. This is emblematic of the wide applicability of spatial data in many avenues of human endeavor. Therefore, efficient storage, retrieval, and processing of spatial data is crucial to deal with the growing need for spatial applications and services.

Technological advances, along with the decreasing costs of storage, computation power, and network bandwidth, have made parallel and distributed processing of large volumes of data attractive. In response to the challenges of Big Spatial Data management, a number of Big Spatial Data systems have emerged in recent years. Many of these Big Spatial Data systems are based on distributed processing of spatial data stored in a distributed file system. These systems vary widely in terms of available features, support for geometry data types and query categories, indexing, data partitioning techniques, and spatial analysis functionalities. For an enterprise or a research organization that is looking for a Big Spatial Data system, these are the key considerations. Performance and scalability are among the most important factors when assessing these systems. A few previous works examined some of these aspects while evaluating Big Spatial Data systems. For instance, Francisco et al. [13] compared two systems with the *Distance Join* operation. Recently, Stefan et al. [16] evaluated three systems with a spatial join operation (involving two point datasets and an Equal predicate) and a range query. However, to the best of our knowledge, none of these projects conducted a comprehensive study of the performance and scalability of Big Spatial Data systems. Therefore, a key goal of this paper is to perform a thorough evaluation of these systems with a comprehensive set of spatial join operations involving the topological relations defined by The Open Geospatial Consortium (OGC) [21] and a series of range queries. In addition, given the importance of spatial analytics, we aim to evaluate a collection of spatial analysis functions. Taking inspiration from our previous work, Jackpine [24], a benchmark to evaluate spatial databases, we have developed a benchmark to evaluate Big Spatial Data systems. In addition to the relevant operations in the Jackpine micro benchmark, in this evaluation study, we have incorporated several new operations. We believe that our benchmark will be helpful to the community of spatial data systems researchers in furthering the state of the art.

In this study, we intend to analyze and compare the features and performance of several Big Spatial Data systems proposed or developed in the last few years. Based on the current trends in Big Data systems, our main focus is on two categories of systems:

Hadoop-based Big Spatial Data systems, which are based on Apache Hadoop [15, 26] framework, and *In-Memory* Spatial Data systems that primarily include systems based on Apache Spark [10, 36]. Hadoop performs its distributed processing of tasks using MapReduce [5] programming paradigm. Hadoop does not have any native support for spatial data processing. Systems like HadoopGIS [1] and SpatialHadoop [8] developed spatial support for Hadoop. Since Hadoop is disk-based and optimized for I/O efficiency, the performance of these systems can deteriorate at scale.

Spark can take advantage of a large pool of memory available in a cluster of machines to achieve better performance rather than disk-based systems. In recent years, several Spark-based spatial data processing systems have been proposed and developed, such as SpatialSpark [33], GeoSpark [19], LocationSpark [20], Simba [6], and STARK [29]. However, there is a huge room for improvement in the performance of these systems, especially in the area of query optimization and efficient Spatial SQL. For the purpose of our evaluation study, we choose two representative systems from the Hadoop-based and Spark-based systems, GeoSpark and SpatialHadoop, since they are the most mature and active projects in each category.

Although Spark is a highly popular framework, it does not offer the best performance due to the overhead associated with scheduling, distributed coordination, and data movement. Researchers [12] have demonstrated that handwritten programs implemented with high performance tools, computational models and scalable algorithms can be orders of magnitude faster than a similar application written with Spark. Consequently, Big Spatial Data systems based on Spark inherit its limitations. To address this issue, recently an APGAS-based distributed in-memory spatio-temporal data system [22] called DISTIL was proposed. However, it only supports point data type and spatio-temporal range queries. There are other non-Spark distributed in-memory big data processing systems, such as Apache Ignite [11], which can also offer better performance than Spark. Apache Ignite supports some important features of big data systems, as well as traditional relational database systems. However, its spatial support is limited to geometry data types (point, line, and polygon) and a limited form of querying on geometry. We propose *SpatialIgnite* that extends Apache Ignite. SpatialIgnite supports spatial join with all OGC [21] compliant topological relations, in addition to range queries and various spatial analysis functions. In this study, we evaluate the performance of SpatialIgnite against two other Big Spatial Data systems: GeoSpark and SpatialHadoop. We demonstrate that SpatialIgnite performs better than both Hadoop-based and Spark-based approaches on real-world spatial datasets.

The main contributions of this paper are as follows:

• We introduce a Big Spatial Data systems benchmark. We also extensively analyze the features and functionalities of each mentioned category of big spatial data systems.

• We present SpatialIgnite, a distributed in-memory Big Spatial Data system based on Apache Ignite. Using our benchmark we conduct a comprehensive performance study of SpatialIgnite, along with SpatialHadoop and GeoSpark.

• As SpatialHadoop and GeoSpark do not support many of the spatial join predicates in their current implementation, we have implemented these features with these systems as part of the evaluation.

The rest of the paper is organized as follows. In Section 2, we review the previous work related to benchmarking Big Spatial Data systems. We present a comparative analysis of existing Big Spatial Data systems in Sections 3, and 4. We describe the benchmark workload in Section 5. We present the performance evaluation in Section 6. Finally, we conclude the paper in Section 7.

2 RELATED WORK

A benchmark plays an important role in evaluating the functionality and performance of a particular system against a reference system. The Transaction Processing Performance Council (TPC) [32] is an organization that has developed several database related benchmarks. Among them, TPC-C (an On-line Transaction Processing benchmark), and TPC-H (a decision support benchmark) are the most widely used. However, TPC does not have any spatial benchmark. Perhaps the earliest known benchmark for spatial databases is SEQUOIA 2000 [30], which focused on raster data based on Earth sciences. Subsequently, OGC played an important role in standardizing spatial topological relations and spatial functions. Jackpine [24] is a popular spatial database benchmark that incorporates a comprehensive workload based on OGC standards as part of its micro benchmark. It also includes a number of real-world applications in its macro benchmark suite. Jackpine was primarily developed to evaluate relational databases with support for spatial functionalities.

Due to the rapid rise in spatial data volume, a number of Big Spatial Data systems have emerged in recent times. It has been demonstrated in [27] that spatial data processing has significantly different characteristics than regular data processing. Therefore, understanding the performance characteristics of these systems is of great interest to many stakeholders and researchers. However, there have been only a few projects (discussed below) that evaluated isolated spatial operations.

Francisco et al. [13] performed a comparative analysis of disk-based spatial data system SpatialHadoop [8] and Spark-based in-memory spatial data system LocationSpark [20]. Their main focus was only *Distance Join* queries. Their analysis shows that LocationSpark performs better than SpatialHadoop in terms of execution time, but SpatialHadoop is a more mature system.

Stefan et al. [16] also analyzed the features and performance of Hadoop and Spark-based Big Spatial data processing systems. They evaluated the performance of their system STARK [29] with SpatialSpark [33], GeoSpark [19] and SpatialHadoop [8] for range queries and a spatial join operation (involving two point datasets and Equal predicate).

Rakesh et al. [23] discussed the architectural comparison of two spatial big data systems SpatialHadoop [8] and GeoSpark [19], where they show that GeoSpark is faster in processing spatial data than SpatialHadoop.

As indexing is one of the most important aspects of spatial data processing, George et al. [25] have done a performance study on Quadtree-based index structure xBR$^+$-tree and R-tree based index structure R*-tree and R$^+$-tree in the context of most common spatial queries, such as point location, window, distance range, nearest-neighbor, and distance-based join. They evaluate the performance based on I/O efficiency and execution time on point dataset and the result shows that the performance of xBR$^+$-tree is better than R*-tree and R$^+$-tree in most cases.

None of the above-mentioned projects conducted a comprehensive performance study of Big Spatial Data systems based on the OGC standards. Hence, there is a great need for such a benchmark that can help the research community for assessing how to take their research to the next level. Our benchmark is intended to fill this void and is inspired by Jackpine. To our knowledge, this is the first comprehensive study of Big Spatial Data systems.

In Sections 3 and 4, we provide a detailed background and feature analysis of Hadoop-based Big Spatial Data systems and distributed in-memory Big Spatial Data systems.

3 HADOOP-BASED BIG SPATIAL DATA SYSTEMS

As Hadoop [15, 26] became a popular framework to process big data in both research community and industry, a number of extensions to Hadoop were proposed. Distributed Big Spatial Data systems like SpatialHadoop [28] and Hadoop-GIS [1] were developed by extending Hadoop and MapReduce framework. A detailed feature matrix of these systems is presented in Table 1.

Hadoop-GIS [1] is a spatial extension of Hadoop to process large-scale spatial data using MapReduce framework. First, it declusters the data and stores it into HDFS. Then it adds a global index to each tile, which is stored in HDFS and shared across the cluster nodes. Its query engine RESQUE can index the data locally on the fly if required, which is stored in memory for faster query processing. Initially, it supported Hilbert Tree and R*-tree for global and local data indexing. Later, SATO [17] was introduced, which is a spatial data partitioning framework integrated with Hadoop-GIS. SATO supports several partitioning and indexing strategies, such as fixed-grid, binary-split, Hilbert-curve, strip, optimized strip, and STR. It can choose an optimal strategy during spatial data processing. Finally, it integrates Hive [3] and extends the HiveQL to support declarative spatial query language. The main issue with Hadoop-GIS is that it is added as a layer on top of Hadoop without changing its system core. As a result, its performance is not improved significantly. In addition, Hadoop-GIS extends Hive for declarative spatial query support, which adds an extra layer of overhead over Hadoop to process spatial queries.

SpatialHadoop [8] is a framework, which incorporates spatial data processing support in different layers of Hadoop, namely, *Storage*, *MapReduce*, *Operations*, and *Language* layers. In the storage layer, SpatialHadoop added a two-level index structure called global and local index. The global index is created for each data partition across the cluster and the local index organizes the data inside each node. Thus, during a query operation, SpatialHadoop can utilize the information regarding which partition is mapped to which node and which block of that node is relevant. This can make the query processing faster. In addition, steps are taken during partitioning to reduce the partition and query skew. It introduces two components in the MapReduce layer, namely, *SpatialFIleSplitter* and *SpatialRecordReader*. SpatialFIleSplitter utilizes the global index to split the input into files and SpatialRecordReader extracts the records from each split by utilizing local index and passing them to the map function. Spatial operations such as range queries, kNN queries, and spatial join query over geometric objects implemented as MapReduce programs in the Operation layer. An OGC-compliant [21] high-level language Pigeon [7] is added to the language layer. Pigeon is an extension of Pig [4, 9], which includes support for geometry data type, spatial predicates, and various spatial operators to run a spatial query on SpatialHadoop. However, it is not efficient to perform join operations with Pigeon, because it uses the cross product operation for joining, which is very costly. Otherwise, since SpatialHadoop modified the system core of Hadoop, it overcomes some of the limitations of Hadoop-GIS and improves the spatial query performance significantly.

Table 1: Hadoop-based Big Spatial Data Systems

Features	SpatialHadoop	Hadoop-GIS
geometry type	point, line, polygon	point, line, polygon
input-format	WKT	WKT
query language	Pigeon	HiveQL with spatial support
partitioning & indexing	grid, R-tree(STR), R$^+$-tree	SATO (fixed-grid, binary-split, hilbert-curve, strip, optimized strip, STR)
spatial operation	range query, kNN, spatial-join, distance-join	range, kNN, spatial-join
query planing & optimization	partition and query skew	partition and query skew
spatial analytics	convexhull, skyline	no
temporal feature	no	no

4 DISTRIBUTED IN-MEMORY BIG SPATIAL DATA SYSTEMS

In this section, we describe two categories of distributed in-memory big spatial data systems: Spark-based distributed in-memory systems and other distributed in-memory systems.

4.1 Spark-based Big Spatial Data Systems

Currently, Apache Spark [10, 37] is widely used distributed in-memory system for big data processing. It can reduce the execution time significantly compared to MapReduce jobs on Hadoop [15, 26]. However, Spark does not have any support for processing spatial data. It processes spatial data by treating it as a non-spatial data due to the lack of spatial indexing, spatial data skew handling and spatial query optimization [20]. To alleviate these limitations, several Spark-based spatial data analysis systems have been proposed in the last few years. A detailed feature matrix of these systems is presented in the Table 2.

GeoSpark [19] is an in-memory cluster computing framework for processing large-scale spatial data. It adds an extension to Apache Spark to support spatial data types and operations. It introduces four different types of Spatial RDDs (SRDDs) based on Spark RDDs [35], namely, *PointRDD*, *RectangleRDD*, *PolygonRDD* and *LineStringRDD*. It efficiently partitions the SRDD data elements across cluster nodes using *Quad-Tree*, *R-Tree*, *Voronoi diagram* and *Fixed-Grid*. It uses

Table 2: Spark-based Big Spatial Data Systems

Features	SpatialSpark	GeoSpark	Simba	LocationSpark	STARK
geometry type	point, line, polygon	point, line, polygon	point	point, line, polygon	geometry
input-format	WKT	CSV, TSV, WKT, WKB, GeoJSON, Shapefile	CSV, JSON, Parquet	WKT	WKT, Time
query language	no	SQL(2017)	SQL	no	Piglet
partitioning	Fixed-Grid, Binary-Split, & Sort-Tile	R-tree, voronoi diagram, Quadtree	STR Partitioner	Grid & region Quadtree	Fixed Grid, Cost-based Binary Split
indexing	R-tree	R-tree, Quad-tree	R-tree	R-tree, Quadtree, IR-tree)	R-tree (live & persistent)
spatial operations	range query, broadcast join & partitioned join	range query, kNN query, spatial-join, distance-join	range query, kNN query, distance-join, kNN-join	range search, range join, kNN Search, kNN-join	kNN query, spatial-join
optimization	no	no	partition & query-skew	partition & query-skew, communication cost (sFilter)	partition skew
memory management	no	no	no	yes	no
spatial analytics	no	no	no	clustering, skyline, spatio-textual topic summarization	clustering
temporal feature	no	no	no	no	yes

Quad-Tree and *R-Tree* indexing techniques to index data on each node. It executes spatial queries such as *range query*, *kNN query*, and *join query* on big spatial datasets by extending the SRDD layer. The system core of GeoSpark mainly consists of three layers. The *Spark Layer* performs basic Spark operations like data loading into a disk, *Spatial RDD Layer* provides geometrical and spatial object support to Spark RDD, and finally, *Spatial Query Processing Layer* performs the spatial query on Spatial RDD. The main limitation of GeoSpark is that it is developed as a library on top of Spark, not as a part of Spark-core, which is not efficient in order to execute spatial queries. Although initially it did not support SQL queries, recently GeoSpark SQL [34] has been introduced. In addition, GeoSpark does not have any support for handling data and query skew.

SpatialSpark [33] implements several spatial operations on Spark to analyze large-scale spatial data. It supports two spatial join operations, where *broadcast join* is used to join a big data set with a smaller dataset and *partition join* is used to join two big dataset. It can perform spatial range query with/without index. These operations can be performed over geometric objects using spatial predicates: intersect, within, overlap, contains, within_distance or nearest_distance. Data can be partitioned using *Fixed-Grid*, *Binary-Split*, and *Sort-Tile* partitioning techniques and indexing using *R-tree*. However, like GeoSpark, it also implemented the spatial support on top of Spark without modifying Spark core. To our knowledge, there is no plan by the SpatialSpark developers to handle data skew and query optimization.

LocationSpark [20] is an efficient spatial data processing system developed based on Apache Spark. It stores spatial data as a key-value pair, where the key can be any geometry data type such as point, a line-segment, a poly-line, a rectangle, or a polygon and the value type can be specified by the user as a text. It supports a wide range of spatial queries including spatial *range-search*, *range-join*, *kNN-search*, and *kNN-join* query. It also supports few spatial analysis functions, such as spatial *data clustering*, *spatial skyline computation*, and *spatio-textual topic summarization*. The query scheduler of LocationSpark contains an efficient cost model and a query execution plan to mitigate and deal with two types of skew: (1) *data partitioning skew* and (2) *query skew*. Its global index (grid and region Quadtree) partitions the data among various nodes and a local index (an R-tree, a variant of the Quadtree, or an IR-tree) used to index data on each node. Also, LocationSpark added a spatial bloom filter to reduce the communication cost of the global spatial index, which can answer whether a spatial point is contained inside a spatial range. Finally, to efficiently manage main memory, it dynamically caches frequently accessed data into memory, and stores less frequently used data into the disk, reducing the number of I/O operations significantly. It can be used as a library on top of Apache Spark. However, it does not have any spatial query language support like SQL.

Simba [6] is a distributed in-memory analytics engine for spatial data processing, which is developed by extending SparkSQL [2] and DataFrame API. The extension of DataFrame API opens the possibility for Simba to interact with other important Spark tools such as MLlib, GraphX etc. It partitions the data using *STR partitioner* by considering partition size, data locality and load balancing. Like LocationSpark, it also adopts a two level indexing (R-tree) scheme, where the global index helps to prune the irrelevant partitions for a query and local index accelerates the query processing in each partition. Currently, it supports spatial operations over point and rectangle objects, including range query, kNN query, spatial distance, and kNN join. Moreover, its spatial-aware and cost-based optimization to select a good query plan helps to achieve both low latency and high throughput. But it only support points, and hence, it is not possible to perform spatial join over geometric objects, such as, polygon or line with Simba. The experimental results show that Simba outperforms SpatialSpark and GeoSpark on point based operations.

STARK [29] is a spatio-temporal data processing framework which is tightly integrated with Spark in order to support spatial data types and operators. Currently, STARK supports two types of spatial partitioning. The *fixed grid partitioner* applies a grid over the data space, where each grid cell corresponds to one partition. The *cost-based binary split partitioner* generates partition based on the number of contained elements. STARK allows two modes for indexing, where *live index* is built upon execution for each partition, queried according to the current predicate (contains/intersects), and then thrown away. And the *persistent indexing* allows to create the index and write the indexed RDD to disk or HDFS. This stored index can be loaded again to save the cost of generating it. But it can run a query on both indexed as well as unindexed data. STARK also extended the Pig Latin, called Piglet, to support declarative query language for spatial data processing. In Piglet, STARK introduced geometry data type and added spatial operators for spatial predicates, join and indexing. Based on the information available, among the Spark-based spatial data systems, only STARK supports temporal feature, but no evaluation of that feature has been reported.

Table 3: Features of SpatialIgnite

Features	**SpatialIgnite**
geometry type	point, line, polygon
input format	WKT
query language	Distributed SQL
partitioning	Rendezvous Hashing
indexing	R-tree
query planning & optimization	yes
spatial operation	spatial-join (supports all OGC-compliant join predicates), range query
spatial analytics	all (see Table 5)
temporal feature	no

4.2 Other Distributed In-memory Big Spatial Data Systems

Apache Ignite [11] is an open source distributed in-memory big data processing platform, which supports many features of big data systems as well as some features of relational DBMS. Caching and parallel query processing on in-memory data are two important features that contribute to its good performance. First, it keeps data in a cache in the main memory data grid distributed across a cluster of nodes and it is horizontally scalable. Second, it performs parallel processing of queries on data in the cache. In addition, it can execute distributed join based on partitioned parallelism. However, Ignite's support for spatial features is rather limited. Currently, it has support for geometry data types (point, line, and polygon) and a limited form of querying on geometry objects.

We introduce SpatialIgnite, which is an extension of Apache Ignite. We implemented the spatial join and spatial analysis support of SpatialIgnite using the JTS library [18]. In this study, we evaluate the performance of SpatialIgnite with two other existing big spatial data systems. The main features of SpatialIgnite is illustrated in Table 3.

Table 4: Existing Support of Spatial Join Predicates

Join Predicates	SpatialHadoop	GeoSpark	SpatialIgnite
Equals	N	N	**Y**
Intersects	**Y**	**Y**	**Y**
Touches	N	N	**Y**
Overlaps	**Y**	N	**Y**
Contains	N	**Y**	**Y**
Crosses	N	N	**Y**

Table 5: Existing Support of Spatial Analysis Functions

Spatial Analysis	SpatialHadoop	GeoSpark	SpatialIgnite
Distance	N	N	**Y**
Within	N	N	**Y**
Dimension	N	N	**Y**
Envelope	**Y**	N	**Y**
Length	N	N	**Y**
Area	N	N	**Y**
Buffer	N	N	**Y**
ConvexHull	**Y**	N	**Y**

5 OUR BENCHMARK

As mentioned earlier, our goal is to conduct a comprehensive evaluation of Big Spatial Data systems. Our benchmark is inspired by Jackpine [24] micro benchmark, and it includes various spatial join operations with OGC-compliant topological predicates and spatial analysis functions. However, not all of the features are supported by most of the Big Spatial Data system. The existing spatial join predicates and analysis functions supported by SpatialHadoop, and GeoSpark are shown in Tables 4 and 5, where 'Y' means a feature is supported and 'N' means it is not supported. Also, these two tables give the features supported by our SpatialIgnite system.

5.1 Workload

Our benchmark workload is comprised of spatial join operations, range queries and spatial analysis functions. The complete list of the operations in our benchmark are shown in Table 6. We adopted these operations from Jackpine and made some changes (marked with '*'), which are listed in the table. Specifically, in Jackpine benchmark, some pair-wise join operations involving line dataset did not use the whole line dataset (e.g. LineCrossesLine). They were changed to use the entire line dataset. Also, some pair-wise join operations involving polygons contained a self-join in Jackpine, such as the join operation 'PolygonOverlapsPolygon' involved the arealm dataset. These were changed into a join operation with two different datasets, for example, 'ArealmOverlapsAreaWater'. Also, we added range queries in our benchmark, which were not part of Jackpine.

Table 6: Our benchmark: workload

Predicates /Functions	Operation	Description
Topological Relations (all pair joins)		
Equals	Polygon Equals Polygon	Find the polygons that are spatially equal to other polygons in arealm dataset.
Equals	Point Equals Point	Find the points that are spatially equal to other points in pointlm dataset.
Intersects	Point Intersects Polygon	Find the points in point dataset that intersect polygons in arealm dataset.
Intersects	Point Intersects Line	Find the points in point dataset that intersect lines in edges dataset.
Intersects	Line Intersects Polygon	Find the lines in edges dataset that intersect polygons in arealm dataset.
Touches*	Polygon Touches Polygon	Find the polygons in arealm dataset that touches polygons in areawater dataset.
Touches*	Line Touches Polygon	Find the lines in edges dataset that touches polygons in arealm dataset.
Overlaps*	Polygon Overlaps Polygon	Find the polygons in arealm dataset that overlaps with polygons in areawater dataset.
Contains*	Polygon Contains Polygon	Find the polygons in arealm dataset that contains the polygons in areawater dataset.
Within*	Polygon Within Polygon	Find the polygons in areawater dataset that are inside the polygons in arealm dataset.
Within	Point Within Polygon	Find the points in pointlm dataset that are inside the polygons in arealm dataset.
Within	Line Within Polygon	Find the lines in edges dataset that are inside the polygons in arealm dataset.
Crosses	Line Crosses Polygon	Find the lines in edges dataset that crosses polygons in arealm dataset.
Crosses*	Line Crosses Line	Find the lines that crosses other lines in edges dataset.
Spatial Analysis		
ConvexHull	Convex Hull of Points	Construct the convex hulls of all points in pointlm dataset.
Envelope	Envelope of Lines	Find the envelopes of all lines in edges dataset.
Length	Longest Line	Find the longest line in edges dataset.
Area	Largest Area	Find the largest polygon in areawater dataset.
Length	Total Line Length	Determine the total length of all lines in edges dataset.
Area	Total Area	Determine the total area of all polygons in areawater dataset.
Dimension	Dimension of Polygons	Find the dimension of all polygons in arealm dataset.
Buffer	Buffer of Polygons	Construct the buffer regions around one mile radius of all polygons in arealm dataset.
Distance	Distance Search	Find all polygons in arealm dataset that are within 1000 distance units from a given point.
Within	Bounding Box Search	Find all lines in edges dataset that are inside the bounding box of a given specification.
Range Query		
Range Query (Point)		Find all the points in pointlm dataset for a given query window.
Range Query (Polygon)		Find all the polygons in arealm dataset for a given query window.
Range Query (Polygon)		Find all the polygons in areawater dataset for a given query window.
Range Query (Line)		Find all the lines in edge dataset for a given query window.

Table 7: Our benchmark: datasets

Dataset	Geometry	Cardinality	Description
pointlm	Point	49837	represents location points (a airport, a movie theater)
arealm	Polygon	5951	represents boundary areas (a city, a national park)
areawater	Polygon	39334	represents water areas (a lake, a river)
edges	Line	4173498	represents lines (roads, rivers)

5.2 Datasets

Our benchmark utilizes real-world spatial datasets which were obtained from Tiger [31] (2011 release). These datasets consist of points, lines, and polygons of California, USA. We have used four datasets from Tiger, including arealm (polygon), areawater (polygon), pointlm (point) and edges (line). The details of these datasets are given in Table 7.

6 PERFORMANCE EVALUATION

For the evaluation of Big Spatial Data systems, we consider three systems, one from each category of the big spatial data systems. First, we choose a Hadoop-based system SpatialHadoop [8, 28] as it is a mature system and its performance is better than any other systems developed on Hadoop. Second, we include Spark-based in-memory system GeoSpark [19], as it is one of the most active projects (latest version v1.1.3 published on 26th April 2018) [14]. Also, it is mentioned as a Spark-based third-party infrastructure project. Finally, we select SpatialIgnite, which we have proposed.

6.1 Benchmark Implementation

SpatialHadoop [8, 28] implemented the SJMR (Spatial Join with MapReduce) algorithm in the core of SpatialHadoop using spatial predicate intersect/overlap to perform the spatial join operation. It does not implement all the OGC-compliant [21] join predicates inside the core of SpatialHadoop (see Table 4). But they implemented the join predicates as part of a Pigeon [7], which is added into the language layer of Hadoop. However, Pigeon performs join operation through a cross product, which is very costly operation. From

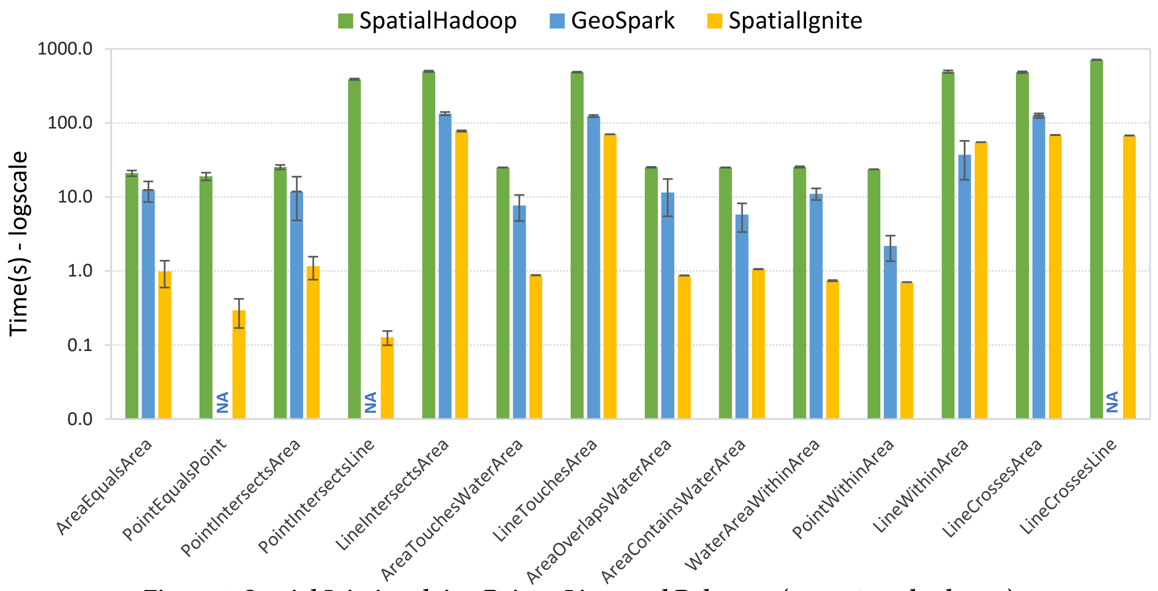

Figure 1: Spatial Join involving Points, Lines and Polygons (on an 8-node cluster)

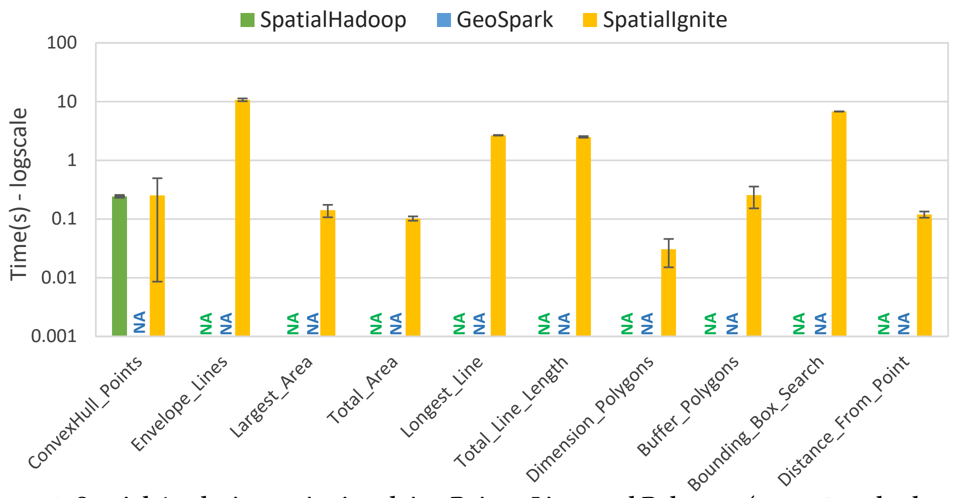

Figure 2: Spatial Analysis queries involving Points, Lines and Polygons (on an 8-node cluster)

our experience, even to perform a join operation like *PointIntersectArea* on a small dataset, *pointlm* and *arealm* take almost 4 hours, whereas it takes only 25 seconds (approx) if we run the same query from SpatialHadoop command line (see Figure 1). We implemented all the spatial join predicates based-on SJMR algorithm. All spatial join queries run using our implemented spatial join predicates in this study.

Similarly, GeoSpark [14, 19] also implemented the spatial join queries using contains/intersects predicates and it can run join queries with or without using index (see Table 4). We implemented other spatial join predicates on GeoSpark and changed the GeoSpark core accordingly to run the spatial join queries.

Finally, we implemented the spatial join and analysis function supports on Apache Ignite using JTS library [18] called SpatialIgnite.

6.2 Experimental Setup

The experiments were conducted on a cluster of 8 machines, each having an Intel(R) Xeon(R) CPU E5472 @ 3.00$GHz \times 2$ with 4×2 cores, 16GB RAM, and 500GB HDD running on Ubuntu 14.04 64-bit operating system with Oracle JDK 1.8.0_81. Along with Hadoop-2.3.0, Spark-2.1.1 and Apache-Ignite-Fabric-1.2.0 used in this evaluation.

6.3 Result Analysis and Discussion

We evaluated three categories of operations, as outlined in Table 6, for performance evaluation. They include pair-wise spatial join, spatial analysis, and range queries. The execution time of a query in each case is reported based on the average elapsed time calculated over several runs.

• **Spatial Join:** We ran 14 types of pairwise spatial join queries involving point, line, and polygon on SpatialHadoop, GeoSpark, and SpatialIgnite. In case of GeoSpark, we were not able to run some queries (marked as 'NA' in Figure 1), because its execution engine is designed in a way such that the query window is always a Polygon. Thus, one dataset is always a polygon during the join operation. If we look at Figure 1, in each case, GeoSpark and SpatialIgnite outperform SpatialHadoop in terms of execution time. Also, SpatialIgnite does better than GeoSpark in all cases, except for the query 'LineWithinArea', where GeoSpark performs better. Overall, the performance of SpatialIgnite is better than the other systems.

• **Spatial Analysis Queries:** SpatialIgnite supports all of the spatial analysis functions in Table 6. However, SpatialHadoop only supports ConvexHull of points. GeoSpark does not have any support for the spatial analysis functions. Figure 2 shows the average execution time of 10 spatial analysis operations involving point, line and polygon dataset. The time to construct the convex hull of all points is almost the same in both SpatialHadoop and SpatialIgnite.

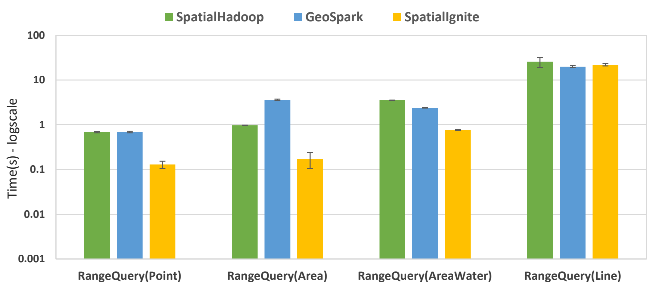

Figure 3: Range queries involving Points, Lines and Polygons (on an 8-node cluster)

• **Range Query:** We ran the range queries (in Table 6) involving point, line and polygon datasets for a given query window. The performance of SpatialHadoop and GeoSpark is almost the same for range queries involving point (pointlm) and polygon (areawater) datasets. However, SpatialHadoop performs better than GeoSpark for range queries in the polygon (arealm) dataset. In each case, SpatialIgnite performs better than the other two, except for line dataset, where the performance of range queries is almost similar with each system. Overall, the performance of SpatialHadoop and GeoSpark is not significantly different in each case.

• **Scalability with increase in number of nodes:** We also investigated the scalability of the evaluated systems by increasing the number of nodes. We used join queries involving the largest dataset (line), because for other queries SpatialIgnite took less than 1 second in most cases when running on an 8-node cluster. If we look at Figure 4, SpatialHadoop performs well with a 4-node. As it is a disk-based system, the I/O overhead increases with the increase in number of nodes. Also, polygon dataset (arealm and areawater) is not large enough to make an impact on the performance with the increased number of nodes in disk-based systems. SpatialIgnite attains the best speedup of 1.92x for 'LineIntersectsArea' join query and its minimum speedup for all cases is 1.84x (LineWithinArea). With 8 nodes, 'LineWithinArea' is the only join operation in which the execution time of GeoSpark is better than that of SpatialIgnite.

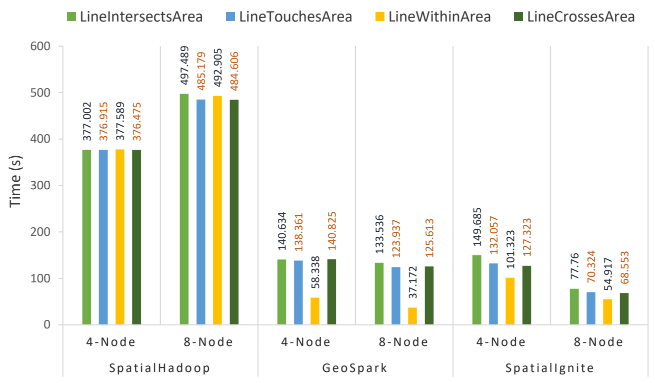

Figure 4: Performance Comparison - Scalability (4-nodes vs 8 nodes)

7 CONCLUSION AND FUTURE WORK

A comprehensive evaluation of Big Spatial Data systems is of great importance to various stakeholders. We have proposed a benchmark and conducted a comprehensive study of Big Spatial Data systems. We investigated all the features and performance issues with these systems, which will help the community in future research of spatial data systems. From our study, we observe that efficient query planning and optimization, memory management, and appropriate selection of data partitioning and indexing technique can improve the performance of Big Spatial Data systems significantly. Furthermore, there is a lack of support for efficient spatial query language like SQL in most of these data systems.

We have also proposed a distributed in-memory spatial data system, called SpatialIgnite, which is an extension of Apache Ignite. Overall, SpatialIgnite performs better than GeoSpark and SpatialHadoop. We have found that GeoSpark returns wrong results in a few queries.

In future, we would like to incorporate support for kNN and distance-join queries within SpatialIgnite. We would also like to evaluate the performance of recent Spark-based systems, such as Simba and LocationSpark. We would also like to evaluate the performance of commercial geospatial systems like GeoMesa and GeoTrellis.

8 ACKNOWLEDGEMENTS

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants of Bhavsar and Ray, and the New Brunswick Innovation Foundation (NBIF) Research Assistantship Initiative Grant (Bhavsar) and an NBIF Start-Up Grant (Ray).

REFERENCES

[1] A. Ablimit, W. Fusheng, V. Hoang, L. Rubao, L. Qiaoling, Z. Xiaodong, and S. Joel. 2013. Hadoop GIS: A High Performance Spatial Data Warehousing System over Mapreduce. *Proc. VLDB Endow.* (2013).

[2] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *ACM SIGMOD.*

[3] T. Ashish, S. J. Sen, J. Namit, S. Zheng, C. Prasad, A. Suresh, L. Hao, W. Pete, and M. Raghotham. 2009. Hive: A Warehousing Solution over a Map-reduce Framework. *Proc. VLDB Endow.* (2009).

[4] O. Christopher, R. Benjamin, S. Utkarsh, K. Ravi, and T. Andrew. 2008. Pig Latin: A Not-so-foreign Language for Data Processing. In *ACM SIGMOD.*

[5] J. Dean and S. Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* (2008).

[6] X. Dong, L. Feifei, Y. Bin, L. Gefei, Z. Liang, and G. Minyi. 2016. Simba: Efficient In-Memory Spatial Analytics. In *ACM SIGMOD*.

[7] A. Eldawy and M. F. Mokbel. 2014. Pigeon: A Spatial MapReduce Language. In *ICDE*.

[8] A. Eldawy and M. F. Mokbel. 2015. SpatialHadoop: A Mapreduce Framework for Spatial Data. In *ICDE*.

[9] The Apache Software Foundation. 2007. "Apache Pig". (2007). https://pig.apache.org/

[10] The Apache Software Foundation. 2013. "Apache Spark". (2013). https://spark.apache.org/

[11] The Apache Software Foundation. 2015. "Apache Ignite". (2015). https://ignite.apache.org/

[12] M. Frank, I. Michael, and M. Derek G. 2015. Scalability! But at What Cost?. In *USENIX HOTOS*.

[13] F. García-García, A. Corral, L. Iribarne, G. Mavrommatis, and M. Vassilakopoulos. 2017. A Comparison of Distributed Spatial Data Management Systems for Processing Distance Join Queries. In *ADBIS*.

[14] geospark 2018. GeoSpark. https://github.com/DataSystemsLab/GeoSpark/releases. (2018).

[15] Hadoop 2018. Apache Hadoop. https://hadoop.apache.org/. (2018).

[16] S. Hagedorn, P. Götze, and Kai-Uwe Sattler. 2017. Big Spatial Data Processing Frameworks: Feature and Performance Evaluation. In *EDBT*.

[17] V. Hoang, A. Ablimit, and W. Fusheng. 2014. SATO: A Spatial Data Partitioning Framework for Scalable Query Processing. In *SIGSPATIAL GIS*.

[18] Java Sun Microsystems Inc. 2017. "JTS Topology Suite". (2017). http://www.tsusiatsoftware.net/jts/main.html

[19] Y. Jia, W. Jinxuan, and S. Mohamed. 2015. GeoSpark: A Cluster Computing Framework for Processing Large-scale Spatial Data. In *SIGSPATIAL GIS*.

[20] T. Mingjie, Y. Yongyang, M. Qutaibah M., O. Mourad, and W. G. Aref. 2016. LocationSpark: A Distributed In-memory Data Management System for Big Spatial Data. *Proc. VLDB Endow.* (2016).

[21] OGC 2018. Open Geospatial Consortium. Simple Feature Access - Part 2: SQL Option. http://www.opengeospatial.org/standards/sfs. (2018).

[22] M. Patrou, M. M. Alam, P. Memarzia, S. Ray, V. C. Bhavsar, K. B. Kent, and G. W. Dueck. 2018. DISTIL: A Distributed In-Memory Data Processing System for Location-Based Services. In *SIGSPATIAL GIS*.

[23] N. Gupta S. M. Ali A. Rath R. K. Lenka, R. K. Barik and H. Dubey. 2016. Comparative Analysis of SpatialHadoop and GeoSpark for Geospatial Big Data Analytics. In *IC3I*.

[24] S. Ray, B. Simion, and A. D. Brown. 2011. Jackpine: A Benchmark to Evaluate Spatial Database Performance. In *ICDE*.

[25] G. Roumelis, M. Vassilakopoulos, A. Corral, and Y. Manolopoulos. 2017. Efficient Query Processing on Large Spatial Databases : A Performance Study. *Journal of Systems and Software* (2017).

[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. 2010. "The Hadoop Distributed File System". In *MSST*.

[27] B. Simion, S. Ray, and A. D. Brown. 2012. Surveying the Landscape: An In-Depth Analysis of Spatial Database Workloads. In *SIGSPATIAL GIS*.

[28] spatialhadoop.cs.umn.edu 2013. SpatialHadoop. https://github.com/aseldawy/spatialhadoop2. (2013).

[29] H. Stefan, G. Philipp, and S. Kai-Uwe. 2017. The STARK Framework for Spatio-Temporal Data Analytics on Spark. *Datenbanksysteme für Business, Technologie und Web* (2017).

[30] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. 1993. The SEQUOIA 2000 storage benchmark. In *ACM SIGMOD*.

[31] Tiger® 2011. http://www.census.gov/geo/www/tiger. (2011).

[32] TPC 1988. The Transaction Processing Performance Council. http://www.tpc.org. (1988).

[33] S. You, J. Zhang, and L. Gruenwald. 2015. Large-Scale Spatial Join Query Processing in Cloud. In *ICDE*.

[34] Y. Chen Z. Huang, L. Wan, and X. Peng. 2017. GeoSpark SQL: An Effective Framework Enabling Spatial Queries on Spark. *ISPRS Int. J. of Geo-Information* (2017).

[35] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *USENIX NSDI*.

[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. 2010. Spark: Cluster Computing with Working Sets. In *USENIX HotCloud*.

[37] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* (2016).