Big Spatial and Spatio-Temporal Data Analytics Systems

Polychronis Velentzas[1], Antonio Corral[2](✉), and Michael Vassilakopoulos[1]

[1] Data Structuring and Engineering Laboratory,
Department of Electrical and Computer Engineering,
University of Thessaly, Volos, Greece
{cvelentzas,mvasilako}@uth.gr

[2] Department of Informatics, University of Almeria, Almeria, Spain
acorral@ual.es

Abstract. We are living in the era of Big Data, and Spatial and Spatio-temporal Data are not an exception. Mobile apps, cars, GPS devices, ships, airplanes, medical devices, IoT devices, etc. are generating explosive amounts of data with spatial and temporal characteristics. Social networking systems also generate and store vast amounts of geo-located information, like geo-located tweets, or captured mobile users' locations. To manage this huge volume of spatial and spatio-temporal data we need parallel and distributed frameworks. For this reason, modeling, storing, querying and analyzing big spatial and spatio-temporal data in distributed environments is an active area for researching with many interesting challenges. In recent years a lot of spatial and spatio-temporal analytics systems have emerged. This paper provides a comparative overview of such systems based on a set of characteristics (data types, indexing, partitioning techniques, distributed processing, query Language, visualization and case-studies of applications). We will present selected systems (the most promising and/or most popular ones), considering their acceptance in the research and advanced applications communities. More specifically, we will present two systems handling spatial data only (SpatialHaddop and GeoSpark) and two systems able to handle spatio-temporal data, too (ST-Hadoop and STARK) and compare their characteristics and capabilities. Moreover, we will also present in brief other recent/emerging spatial and spatio-temporal analytics systems with interesting characteristics. The paper closes with our conclusions arising from our investigation of the rather new, though quite large world of ecosystems supporting management of big spatial and spatio-temporal data.

1 Introduction

We are living in the era of Big Data, and Spatial and Spatio-temporal Data are not an exception. Mobile apps, cars, GPS devices, ships, airplanes, medical devices, IoT devices, etc. are generating explosive amounts of data with spatial and temporal characteristics. Social networking systems also generate and store vast amounts of geo-located information, like geo-located tweets, or captured mobile users' locations. To manage this huge volume of spatial and spatio-temporal data we need

A. Hameurlain et al. (Eds.): TLDKS XLVII, LNCS 12630, pp. 155–180, 2021.
https://doi.org/10.1007/978-3-662-62919-2_7

parallel and distributed frameworks. For this reason, modeling, storing, querying and analyzing big spatial and spatio-temporal data in distributed environments is an active area for researching with many interesting challenges.

In recent years a lot of spatial and spatio-temporal analytics systems have emerged. Parallel and distributed Spatio-temporal analytic systems are mostly either based on Hadoop, or on Spark. Moreover, most of them handle spatial data only, while others can represent and process statio-temporal information. Considering these alternatives, four possible groups of systems are formed: Hadoop or Spark based spatial data systems and Hadoop or Spark based spatio-temporal data systems.

This paper provides a comparative overview of such systems based on a set of characteristics (data types, indexing, partitioning techniques, distributed processing, query Language, visualization and case-studies of applications). We will present selected systems (the most promising and/or most popular ones), considering their acceptance in the research and advanced applications communities.

In Sect. 2 we will introduce parallel and distributed architectures and their two basic representatives (Hadoop and Spark). Next, in Sect. 3 we will present two systems handling spatial data only (SpatialHaddop and GeoSpark) and two systems able to handle spatio-temporal data, too (ST-Hadoop and STARK) and compare their characteristics and capabilities. In Sect. 4, we will present in brief other recent/emerging spatio-temporal analytics systems with interesting characteristics. The paper closes with our conclusions arising from our investigation of the rather new, though quite large world of ecosystems supporting management of big spatial and spatio-temporal data.

2 Parallel and Distributed Architectures

Data mining and analysis of big data is a non-trivial task. Often, it is performed in a distributed infrastructure of multiple compute network-interconnected

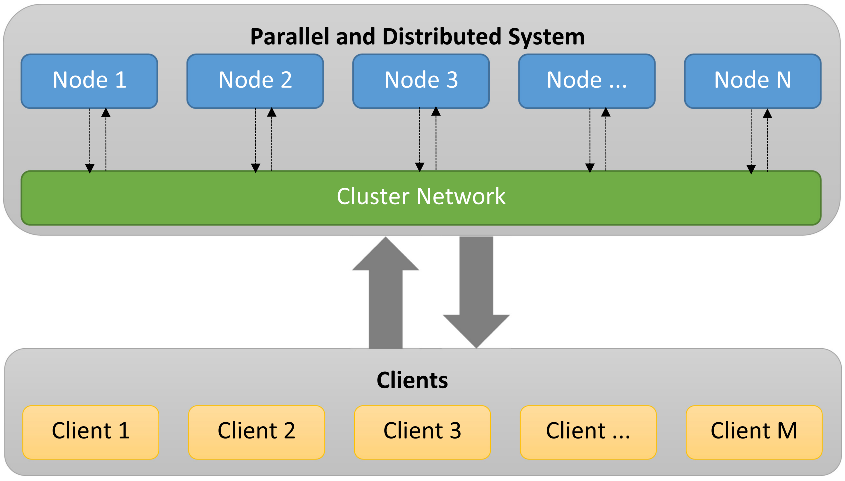


Fig. 1. Parallel and distributed system architecture.

(through the cluster network) nodes, each of which may be equipped with multiple CPUs or GPUs (Fig. 1).

This kind of architectures bring a number of challenges. First of all, the resources must be effectively used. For example, one must avoid delays of CPU/GPU resources due to working data transfer over network. Second, all the available resources (CPU/GPU, storage and network) should be typically shared among different users or processes to reduce costs and increase interoperability. In order to address these challenges several architectures and frameworks have risen. In this section, we will describe the two most popular of them, Apache Hadoop and Apache Spark. Both of them are open source and widely used.

2.1 Apache Hadoop

Hadoop is a shared-nothing framework, meaning that the input data is partitioned and distributed to all computing nodes, which perform calculations on their local data only. Hadoop is a two-stage disk-based MapReduce computation engine, not well suited to repetitive processing tasks.

MapReduce [10,25] is a programming model for distributed computations on very large amounts of data and a framework for large-scale data processing on clusters built from commodity hardware. A task to be performed using the MapReduce framework has to be specified as two phases: a) the *map* phase, which is specified by a *map function*, takes input, typically from Hadoop Distributed File System (HDFS) files, possibly performs some computations on this input, and distributes it to worker nodes, and b) the *reduce* phase which processes these results as specified by a *reduce function* (Fig. 2). An important aspect of MapReduce is that both the input and the output of the *map* step are represented as *key/value pairs* and that pairs with same key will be processed as one group by a *reducer*. The *map* step is parallelly applied to every pair with key k_1 of the input dataset, producing a list of pairs with key k_2. Subsequently, all pairs with the same key from all lists are grouped together, creating one list for each key (*shuffling step*). The *reduce* step is then parallelly applied to each such group, producing a list of key/value pairs:

$$map : (k_1, v_1) \rightarrow list(k_2, v_2) \text{ and } reduce : (k_2, list(v_2)) \rightarrow list((k_3, v_3))$$

Additionally, a *combiner function* can be used to run on the output of the *map* phase and perform some filtering or aggregation to reduce the number of keys passed to the *reducer*. The MapReduce architecture provides good scalability and fault tolerance mechanisms. MapReduce was originally introduced by Google in 2004 and was based on well-known principles of parallel and distributed processing. It has been widely adopted through Hadoop (an open-source implementation), whose development was led by Yahoo and later became an Apache project[1].

[1] https://hadoop.apache.org/.

2.2 Apache Spark

To overcome limitations of the MapReduce paradigm and Apache Hadoop (especially regarding iterative algorithms), Apache Spark[2] was developed. This is also an open-source cluster-computing framework based on Resilient Distributed Datasets (RDDs), read-only multisets of data items distributed over the computing nodes. RDDs form a kind of distributed shared memory, suitable for the implementation of iterative algorithms. Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG (Directed Acyclic Graph) scheduler (an example is depicted in Fig. 3), a query optimizer and a physical execution engine.

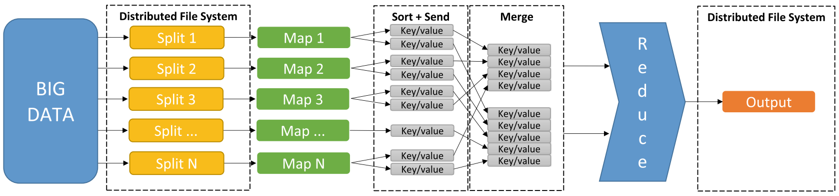


Fig. 2. MapReduce programming model.

DAG scheduler is the scheduling layer of Apache Spark that implements stage-oriented scheduling. It transforms a logical execution plan (i.e. RDD lineage of dependencies built using RDD transformations) to a physical execution plan (using stages).

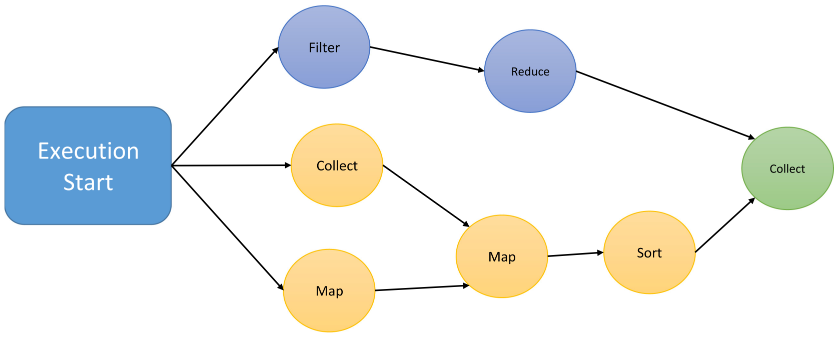


Fig. 3. An example of a DAG (Directed Acyclic Graph) scheduler.

The data transformations that take place in Spark are executed in a "lazy" way. Transformations are lazy in nature: when we call some operation for an

[2] https://spark.apache.org/.

RDD, it does not execute immediately; it is executed when output is requested. Spark maintains a record of which operation is being called (through DAG). We can think of a Spark RDD as the data that we built up through transformations. Since transformations are lazy in nature, we can execute operations any time by calling an action on data. Hence, in lazy evaluation, data is not loaded, and computations are not performed until it is necessary.

3 Big Spatial and Spatio-Temporal Data Analytics Systems

In the next subsections, we will present in detail a popular representative of each of group of systems (Hadoop-based and Spark-based Spatial and Hadoop-based and Spark-based Spatio-temporal Data Analytics Systems). SpatialHadoop (http://spatialhadoop.cs.umn.edu/), a full-fledged MapReduce framework with native support for spatial data, is presented in Subsect. 3.1. Subsect. 3.2 is devoted to GeoSpark (http://geospark.datasyslab.org), an in-memory cluster computing framework for processing large-scale spatial data that uses Spark as its base layer and adds two more layers, the Spatial RDD (SRDD) Layer and Spatial Query Processing Layer, thus providing Spark with in-house spatial capabilities. ST-Hadoop (http://st-hadoop.cs.umn.edu/), the first full-fledged open-source MapReduce framework with a native support for spatio-temporal data, is presented in Subsect. 3.3. ST-Hadoop is a comprehensive extension to Hadoop and SpatialHadoop that injects spatio-temporal data awareness inside each of their layers. In Subsect. 3.4, STARK framework for scalable spatio-temporal data analytics on Spark (https://github.com/dbis-ilm/stark) is presented. It is built on top of Spark and provides a domain specific language (DSL) that seamlessly integrates into any (Scala) Spark program. It includes an expressive set of spatio-temporal operators for filter, join with various predicates as well as k nearest neighbor search. Moreover, in Subsect. 3.5 we present a comparison of these systems regarding their capabilities and characteristics.

3.1 SpatialHadoop

SpatialHadoop (http://spatialhadoop.cs.umn.edu/) [13,14] is a full-fledged MapReduce framework with native support for spatial data. It is an efficient disk-based distributed spatial query processing system. Note that MapReduce [10] is a scalable, flexible and fault-tolerant programming framework for distributed large-scale data analysis.

SpatialHadoop [13,14] (see in Fig. 4 its architecture) is a comprehensive extension to Hadoop [7] that injects spatial data awareness in each Hadoop layer, namely, the language, storage, MapReduce, and operations layers. In the *Language* layer, SpatialHadoop adds a simple and expressive high-level language for spatial data types and operations. In the *Storage* layer, SpatialHadoop adapts traditional spatial index structures as Grid, R-tree, R^+-tree, Quadtree, etc. to form a two-level spatial index [15]. SpatialHadoop enriches the *MapReduce* layer by two new components, *SpatialFileSplitter* and *SpatialRecordReader* for efficient and scalable spatial data processing. *SpatialFileSplitter* (*SFS*) is an extended splitter that exploits the global index(es) on input file(s) to early prune file cells/blocks not contributing to answer, and *SpatialRecordReader* (*SRR*) reads a split originating from spatially indexed input file(s) and exploits the advantages of the local indices to efficiently process it. At the *Operations* layer, SpatialHadoop is also equipped with a several spatial operations, including range query, *k*NN query and spatial join. Other computational geometry algorithms (e.g. polygon union, skyline, convex hull, farthest pair and closest pair) are also implemented following a similar approach [11].

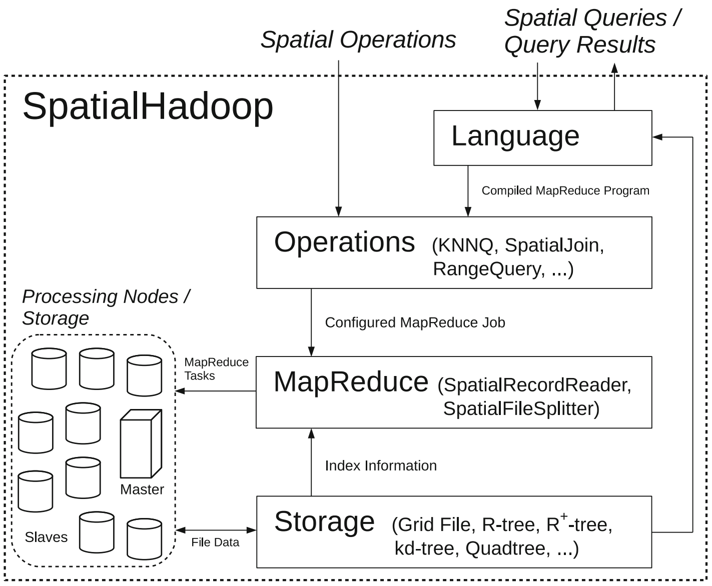

Spatial Operations
Spatial Queries / Query Results
SpatialHadoop
Language
Compiled MapReduce Program
Operations (KNNQ, SpatialJoin, RangeQuery, ...)
Processing Nodes / Storage
Configured MapReduce Job
MapReduce Tasks
MapReduce (SpatialRecordReader, SpatialFileSplitter)
Master
Index Information
Slaves
File Data
Storage (Grid File, R-tree, R+-tree, kd-tree, Quadtree, ...)

Fig. 4. SpatialHadoop system architecture [14].

Spatial Data Types. The *Language* layer provides a high-level language with standard spatial data types and operations to make the system accessible to non-technical users. In particular, the language layer provides Pigeon [12] a simple high level SQL-like language that supports OGC-compliant spatial data types and spatial operations. Pigeon overrides the `bytearray` data type to support standard spatial data types, such as, `Point`, `LineString`, and `Polygon`. Conversion between `bytearray` and `geometry`, back and forth, is done automatically on the fly which makes it transparent to end users.

Spatial Storage (indexing Techniques). SpatialHadoop proposes a two-layer spatial index structure which consists of one global index and multiple local indexes. The *global index* partitions data into HDFS blocks and distributes them among cluster nodes, while *local indexes* organize records inside each block. The separation of global and local indexes lends itself to the MapReduce programming paradigm where the global index is used while preparing the MapReduce job while the local indexes are used for processing the map tasks. In addition, breaking the file into smaller partitions allows each partition to be indexed separately in memory and dumping it to a file in a sequential manner. SpatialHadoop uses this two-level design to build a grid index, R-tree and R^+-tree. The index is constructed in one MapReduce job that runs in three phases. (1) The *partitioning* phase divides the space into n rectangles, then, it partitions the data by assigning each record to overlapping rectangles. (2) In the *local indexing* phase, each partition is processed separately on a single machine and a local index is constructed in memory before it is dumped to disk. (3) The final *global indexing* phase constructs a global index on the master node which indexes all HDFS blocks in the file using their MBRs as indexing key.

Spatial Partitioning Techniques. In [15], seven different *spatial partitioning techniques* in SpatialHadoop are presented, and an extensive experimental study on the quality of the generated index and the performance of range and spatial join queries is reported. These seven partitioning techniques are also classified in two categories according to boundary object handling: *replication-based techniques* (Grid, Quadtree, STR+ and *k*-d tree) and *distribution-based techniques* (STR, Z-Curve and Hilbert-Curve). The *distribution-based techniques* assign an object to exactly one overlapping cell and the cell has to be expanded to enclose all contained points. The *replication-based techniques* avoid expanding cells by replicating each point to all overlapping cells, but the query processor has to employ a duplicate avoidance technique to account for replicated elements. The most important conclusions of [15] for distributed join processing, using the *overlap* spatial predicate, are the following: (1) the smallest running time is obtained when the same partitioning technique is used in both datasets for the join processing, (2) Quadtree outperforms all other techniques with respect to running time, since it minimizes the number of overlapping partitions between the two files by employing a regular space partitioning, (3) Z-Curve reports the worst running times, and (4) *k*-d tree gets very similar results to STR.

Spatial Operations. SpatialHadoop contains a number of basic spatial operations such as range query, kNN query and spatial join [14]. A *range query* takes a set of spatial records P and a query area A as input, and returns the records from P that overlap with A. SpatialHadoop exploits the global index with the SpatialFileSplitter to select only the partitions that overlap the query range A. Then, it uses the SpatialRecordReader to process the local indexes in matching partitions and find matching records. Finally, a duplicate avoidance step filters out duplicate results caused by replication in the index. A *kNN query* algorithm in SpatialHadoop is composed of the three steps: (1) *Initial Answer*, where we come up with an initial answer of the k closest points to the query point q within the same file partition as q. It first locates the partition that includes q by feeding the SpatialFileSplitter with a filter function that selects only the overlapping partition. Then, the selected partition goes through the SpatialRecordReader to exploit its local index with a traditional kNN algorithm to produce the initial k answers. (2) *Correctness check*, where it checks if the initial answer can be considered final or not. (3) *Answer Refinement*, if the correctness check result is not final, a range query is executed to produce the nearest k point as the final result. For a *spatial join query*, SpatialHadoop proposes a MapReduce-based algorithm where the SpatialFileSplitter exploits the two global indexes to find overlapping pair of partitions. The map function uses the SpatialRecordReader to exploit the two local indexes in each pair to find matching records. Finally, a duplicate avoidance step eliminates duplicate pairs in the answer caused by replication in the index. Finally, CG_Hadoop [11] is a suite of computational geometry operations for MapReduce. It supports five fundamental computational geometry operations, namely, polygon union, skyline, convex hull, farthest pair, and closest pair, all implemented as MapReduce algorithms.

Distributed Processing (MapReduce and Dataflow). In general, a spatial query processing in SpatialHadoop consists of four steps [14,19,20], regardless of whether we have one or two input files (see Fig. 5, where two files as input are shown): (1) *Preprocessing*, where the data is partitioned according to a specific spatial index, generating a set of partitions or cells. (2) *Pruning*, when the query is issued, where the master node examines all partitions and prunes by a *filter function* those ones that are guaranteed not to be included in any possible result of the spatial query. (3) *Local Spatial Query Processing*, where a local spatial query processing is performed on each non-pruned partition in parallel on different machines. (4) *Global Processing*, where the results are collected from all machines in the previous step and the final result of the concerned spatial query is computed. A *combine* function can be applied in order to decrease the volume of data that is sent from the *map* task. The *reduce* function can be omitted when the results from the *map* phase are final.

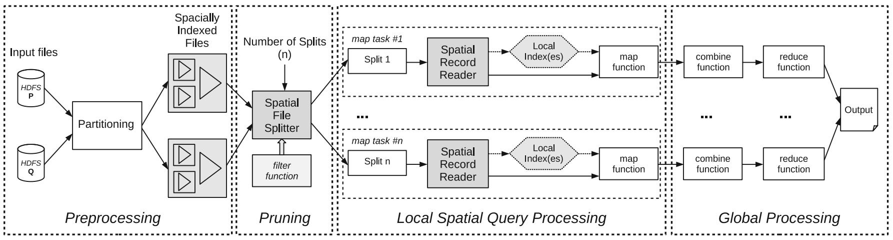


Fig. 5. Spatial query processing in SpatialHadoop [14,20].

Query Language. The *Language* layer contains Pigeon [12], a high level language with OGC-compliant spatial data types and functions. In particular, it adds the following: (1) OGC-compliant spatial data types including, `Point`, `LineString`, `Polygon`. (2) OGC-standard spatial predicates which return a Boolean value based on a test on the input polygon(s); e.g., `IsClosed`, `Touches`. (3) Basic spatial functions which are used to extract useful information from a single shape; e.g., `Area`. (4) Spatial analysis functions which perform some spatial transformations on input objects; e.g., `Centroid`, `Intersection`. (5) Spatial aggregate functions which take a set of spatial objects and return a single value which summarizes all input objects; e.g., `ConvexHull`. (6) and some changes to the language; e.g. kNN Keyword, FILTER, JOIN.

Visualization. The visualization process involves creating an image that describes an input dataset. This is a natural way to explore spatial datasets as it allows users to find interesting patterns in the input which are otherwise hard to spot. SpatialHadoop provides a visualization layer which generates two types of images, namely, *single level* image and *multilevel* images. For *single level* image visualization, the input dataset is visualized as a single image of a user-specified image size (width x height) in pixels. SpatialHadoop generates a single level image in three phases. (1) *Partitioning* phase partitions the data using either the default non-spatial Hadoop partitioner or using the spatial partitioner in SpatialHadoop depending on whether the data needs to be smoothed or not. (2) In the *Rasterize* phase, the machines in the cluster process the partitions in parallel and generate a partial image for each partition. (3) In the *Merging* phase, the partial images are combined together to produce the final image. SpatialHadoop also supports *multilevel* images which consist of small tiles produced at different zoom levels. SpatialHadoop provides an efficient algorithm that runs in two phases, *partition* and *rasterize*. (1) The *Partition* phase scans all input records and replicates each record r to all overlapping tiles in the image according to the MBR of r and the MBR of each tile. This phase produces one partition per tile in the desired image. (2) The *Rasterize* phase processes all generated partitions and generates a single image out of each partition.

Case-Studies of Applications. The core of SpatialHadoop is used in several real applications that deal with big spatial data including MNTG [30], a web-based traffic generator; TAREEG [2], a MapReduce extractor for OpenStreetMap data; TAGHREED [29], a system for querying and visualizing twitter data, and SHAHED [16], a MapReduce system for analyzing and visualizing satellite data. SHAHED is a tool for analyzing and exploring remote sensing data publicly available by NASA in a 500 TB archive. It provides a web interface where users navigate through the map and the system displays satellite data for the selected area. HadoopViz [17] is a MapReduce-based framework for visualizing big spatial data, it can efficiently produce giga-pixel images for billions of input records.

3.2 GeoSpark

The **GeoSpark** (http://geospark.datasyslab.org) framework exploits the core engine of Apache Spark and SparkSQL, by adding support for spatial data types, indexes, and geometrical operations. GeoSpark extends the Resilient Distributed Datasets (RDDs) concept to support spatial data. It adds two more layers, the Spatial RDD (SRDD) Layer and Spatial Query Processing Layer, thus providing Spark with in-house spatial capabilities. The SRDD layer consists of three newly defined RDDs, PointRDD, RectangleRDD and PolygonRDD. SRDDs support geometrical operations, like Overlap and Minimum Bounding Rectangle. SRDDs are automatically partitioned by using the uniform grid technique, where the global grid file is split into a number of equal geographical size grid cells. Elements that intersect with two or more grid cells are being duplicated. GeoSpark provides spatial indexes like Quadtree and R-tree on a per partition base. The Spatial Query Processing Layer includes spatial range query, spatial join query, spatial *k*NN query. GeoSpark relies heavily on the JTS (Java Topology Suite) and therefore conforms to the specifications published by the Open Geospatial Consortium. It is a robust and well implemented spatial system. Moreover, a lot of heterogeneous data sources are supported, like CSV, GeoJSON, WKT, NetCDF/HDF and ESRI Shapefile. GeoSpark does not directly support temporal data and operations.

Spatial and Spatio-Temporal Data Types. All the common spatial datatypes are supported like Point, Multi-Point, Polygon, Multi-Polygon, LineString, Multi-LineString, GeometryCollection, and Circle. In addition to these simple datatypes, GeoSpark is integrated with the complex geometrical shapes concave/convex polygons and multiplesub-shapes.

Spatial and Spatio-Temporal Storage (Indexing Techniques). The framework indexing architecture is built as a set of indexes per RDD partition. GeoSpark spatial indexes rely on the R-tree or Quadtree data structure. There are three kind of index options:

1. Build local indexes: GeoSpark builds a set of indexes per spatial RDD. This way a global index is not created and all objects are not indexed in one machine. Furthermore, to speedup queries the indexes are clustered indexes, meaning that spatial objects are stored directly in the spatial index. As a result, querying the index returns immediately the spatial object, skipping the I/O overhead of a second retrieve based on the objects pointer.
2. Query local indexes: The queries are divided in smaller tasks and these tasks are executed in parallel. The framework will use any existing local spatial indexes, minimizing query execution time.
3. Persist local indexes: GeoSpark users have the option to reuse the build local indexes, by storing it in one of the following ways: (1) cache to memory by calling IndexedSpatialRDD.cache(), (2) persist on disk by calling IndexedSpatialRDD.saveAsObjectFile(HDFS/S3 PATH).

Spatial and Spatio-Temporal Partitioning Techniques. In order to take advantage of the spatial proximity which is crucial for improving query speed, GeoSpark automatically repartitions a loaded Spatial RDD according to its internal spatial data distribution (Fig. 6 presents spatial partitioning techniques supported by GeoSpark). This is crucial for every computation, because it minimizes the data shuffles across the cluster and it avoids unnecessary CPU overheads on partitions that contain unwanted data. The framework implements spatial partitioning in three main steps:

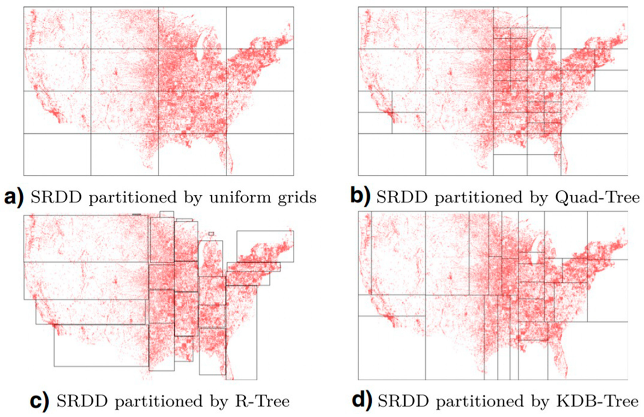

a) SRDD partitioned by uniform grids
b) SRDD partitioned by Quad-Tree
c) SRDD partitioned by R-Tree
d) SRDD partitioned by KDB-Tree

Fig. 6. Spatial partitioning techniques [42].

1. Building a global spatial grid file: Each Spatial RDD partition is sampled and the data are collected by the master node, resulting to a small subset of the spatial RDD. The sampled RDD is divided in equally load balanced partitions and their boundaries are used to further partition the initial RDD, resulting to new spatial RDD partitions, which are also load balanced. GeoSpark offers the following partition options: Uniform Grid, R-tree, Quadtree and kDB-tree.
2. Assigning a grid cell ID to each object: After constructing the global grid file, the framework assigns a grid cell to each object. Therefore, it creates a new spatial RDD whose schema is <Key, Value>. Every spatial object is stored in the new RDD with its corresponding grid cell ID. In case a spatial object span across multiple grids, the spatial RDD may contain duplicates.
3. Re-partitioning SRDD across the cluster: The Spatial RDD generated by the last step has a <Key, Value> pair schema. The Key represents a grid cell ID. In this the spatial RDD is repartitioned by the key, and the objects with the same key are grouped into the same partition.

Spatial and Spatio-Temporal Operations. The GeoSpark framework comes with a full range of spatial operations. Users can facilitate spatial analysis and spatial data mining by combining queries with one or more of the following spatial operations:

1. Spatial Range query: This operation returns all the spatial objects that lie within a defined region. As an example, this query can find all gas stations in the city center.
2. Spatial join: This kind of queries combine two or more datasets, using spatial predicates (e.g. Intersects, Overlaps, Contains, Distance etc).
3. Spatial k nearest neighbors (kNN) query: kNN query computes the k nearest neighbors around a center point. For example, a kNN query could be, find the 5 nearest hotels around the user.

Distributed Processing (MapReduce and Dataflow). GeoSpark can run spatial query processing operations on the SRDDs, right after the Spatial RDD layer loads, partitions are generated and indexing is completed. The spatial query processing layer provides support for many spatial operations like range query, distance query, k Nearest Neighbors (kNN) query, range join query and distance join query. In order to describe the distributed processing of GeoSpark we will analyze the simplest of the queries the range query. A spatial range query is faster and less resource-consuming because it just returns objects that the input query window object contains. To complete such queries, we need to issue a parallelized Filter transformation in Apache Spark, which introduces a narrow dependency. As a result, repartitioning is not needed. These is also a more efficient way, we can broadcast the query window to all workers and parallelize the processing across the cluster. The query processing algorithm needs only one stage, due to the narrow dependency which does not require data shuffle. In Fig. 7, the range query DAG and data flow is depicted.

Query Language. GeoSparkSQL supports SQL/MM Part3 Spatial SQL Standard. It includes four kinds of SQL operators as follows. All these operators can be directly called through this command in Scala: var myDataFrame = sparkSession.sql("YOUR SQL HERE").

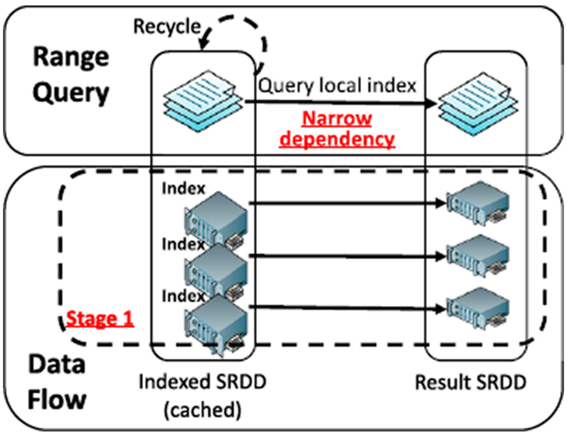


Fig. 7. Range query DAG and data flow [42].

1. Constructor: Constructor creates a geometry from an input string or coordinates. For example, we have the following constructor ST_GeomFromWKT (string), which constructs a Geometry from Wkt, ST_GeomFromGeoJSON (string) which constructs a Geometry from a JSON string, ST_Point (decimal, decimal) which constructs a Point from coordinates,
2. Function: There are many available functions like ST_Distance (geometry, geometry) which returns the Euclidean distance between two geometries, ST_Area (geometry) that calculates the area of a geometry and many more.
3. Predicate: The spatial predicates describe the spatial relationships. They also imply a spatial logic amongst the spatial objects which is essentially a spatial join. GeoSpark supports a complete set of predicates like ST_Contains(geometry, geometry), ST_Intersects (geometry, geometry), ST_Equals (geometry, geometry).
4. Aggregate function: SQL has aggregate functions, which are used to aggregate the results of a SQL query. Likewise, GeoSparkSQL also supports spatial aggregate functions. Spatial aggregate functions aggregate the results of SQL queries involving geometry objects. For example, ST_Union_Aggr(geometryColumn) returns the polygon union of all polygons of the geometryColumn.

Visualization. GeoSpark visualization is supported with the core visualization framework GeoSparkViz [41]. GeoSparkViz a large-scale geospatial map visualization framework. GeoSparkViz extends Apache Spark with native support for general cartographic design. It offers a plethora of utilities that enable users to perform data management and visualization on spatial data. One of the best features of GeoSparkViz is that it reduces the overhead of loading the intermediate spatial data generated during the data management phase to the designated map visualization tool.

3.3 ST-Hadoop

ST-Hadoop (http://st-hadoop.cs.umn.edu/) [4,6], see in Fig. 8 its architecture, is a full-fledged open-source MapReduce framework with a native support for spatio-temporal data. ST-Hadoop is a comprehensive extension to Hadoop [7] and SpatialHadoop [14] that injects spatio-temporal data awareness inside each of their layers, mainly, language, indexing, MapReduce and operations layers. In the *Language* layer, ST-Hadoop extends Pigeon language [12] to supports spatio-temporal data types and operations. In the *Indexing* layer, ST-Hadoop spatio-temporally loads and divides data across computation nodes in the Hadoop Distributed File System (HDFS). In this layer, ST-Hadoop scans a random sample obtained from the whole dataset, bulk loads its spatio-temporal index in-memory, and then uses the spatio-temporal boundaries of its index structure to assign data records with its overlap partitions. ST-Hadoop sacrifices storage to achieve more efficient performance in supporting spatio-temporal operations, by replicating its index into temporal hierarchy index structure that consists of two-layer indexing of temporal and then spatial. The *MapReduce* layer introduces two new components of *SpatioTemporalFileSplitter* and *SpatioTemporalRecordReader*, that exploit the spatio-temporal index structures to speed up spatio-temporal operations. Finally, the *Operations* layer encapsulates the spatio-temporal operations that take advantage of the ST-Hadoop temporal hierarchy index structure in the indexing layer, such as spatio-temporal range, spatio-temporal top-k nearest neighbor, and spatio-temporal join queries.

The key idea behind the performance gain of ST-Hadoop is its ability to load the data in HDFS in a way that mimics spatio-temporal index structures [3]. Hence, incoming spatio-temporal queries can have minimal data access to retrieve the query answer. The extensibility of ST-Hadoop allows others to extend spatio-temporal features and operations easily using similar approaches as described in [6].

Spatial and Spatio-Temporal Data Types. Spatio-temporal data types (STPoint, Time and Interval) are used to define the schema of input files upon their loading process. ST-Hadoop extends `STPoint`, `TIME` and `INTERVAL`. For instance, the `TIME` instance is used to identify the temporal dimension of the data, while the time `INTERVAL` mainly provided to equip the query predicates.

Spatial and Spatio-Temporal Storage (Indexing Techniques). ST-Hadoop HDFS organizes input files as spatio-temporal partitions that satisfy one main goal of supporting spatio-temporal queries. ST-Hadoop imposes *temporal slicing*, where input files are spatio-temporally loaded into intervals of a specific time granularity, e.g., days, weeks, or months. Each granularity is represented as a level in ST-Hadoop index. Data records in each level are spatio-temporally partitioned, such that the boundary of a partition is defined by a spatial region and time interval.

The key idea behind the performance gain of ST-Hadoop is its ability to load the data in HDFS in a way that mimics spatio-temporal index structures. To support all spatio-temporal operations including more sophisticated queries over time, ST-Hadoop replicates spatio-temporal data into a *Temporal Hierarchy Index*. ST-Hadoop set *Temporal Hierarchy Index* structure to four levels of days, weeks, months and years granularities, but it can be changed by the users.

ST-Hadoop index structure consists of two-layer indexing of a temporal and spatial. This two-layer indexing is replicated in all levels, where in each level the sample is partitioned using different granularity. ST-Hadoop trade-off storage to achieve more efficient performance through its index replication. In general, the index creation of a single level in the *Temporal Hierarchy* goes through four consecutive phases, called sampling, temporal slicing, spatial indexing, and physical writing. For instance, in the spatial indexing phase, ST-Hadoop determines the spatial boundaries of the data records within each temporal slice. ST-Hadoop spatially index each temporal slice independently, and it takes the advantages of applying different types of spatial bulk loading techniques in HDFS that are

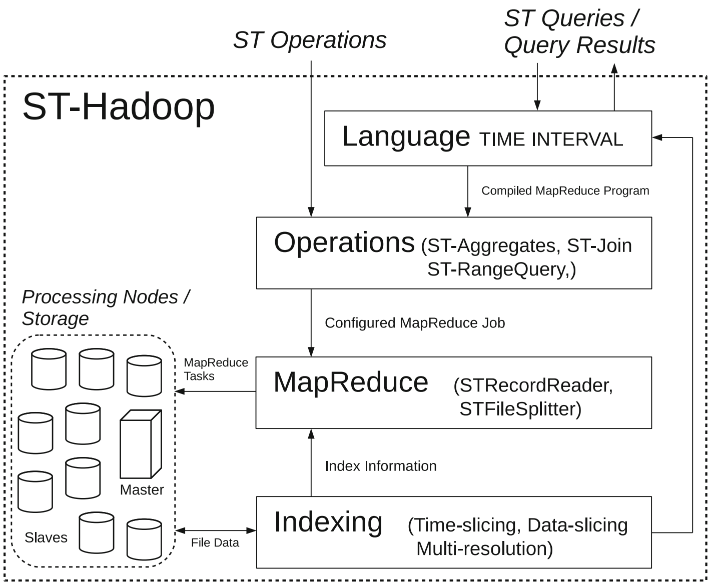

ST Operations
ST Queries / Query Results
ST-Hadoop
Language TIME INTERVAL
Compiled MapReduce Program
Operations (ST-Aggregates, ST-Join ST-RangeQuery,)
Configured MapReduce Job
Processing Nodes / Storage
MapReduce Tasks
MapReduce (STRecordReader, STFileSplitter)
Index Information
Master
Slaves
File Data
Indexing (Time-slicing, Data-slicing Multi-resolution)

Fig. 8. ST-Hadoop system architecture [4,6].

already implemented in SpatialHadoop (Grid, STR-tree, Quadtree and k-d tree). The output of this phase is the spatio-temporal boundaries of each temporal slice.

Spatial and Spatio-Temporal Partitioning Techniques. In the temporal slicing phase, ST-Hadoop determines the temporal boundaries by slicing the in-memory sample into multiple time intervals, to efficiently support a fast-random access to a sequence of objects bounded by the same time interval. ST-Hadoop employs two *temporal slicing* techniques, where each manipulates the sample according to specific slicing characteristics: (1) *Time-partition slicing*, slices the sample (from the sampling phase) into multiple splits that are uniformly on their time intervals, and (2) *Data-partition slicing* where the sample is sliced to the degree that all sub-splits are uniformly in their data size. The output of the temporal slicing phase finds the temporal boundary of each split, that collectively cover the whole time domain. Moreover, ST-Hadoop takes the advantages of applying different types of spatial bulk loading techniques in HDFS that are already implemented in SpatialHadoop such as Grid, STR-tree, Quadtree and k-d tree.

Spatial and Spatio-Temporal Operations. The combination of the spatio-temporally load balancing with the temporal hierarchy index structure gives the kernel of ST-Hadoop, that enables the possibility of efficient and practical realization of spatio-temporal operations. The *Operations* layer encapsulates the implementation of three common spatio-temporal operations, namely, spatio-temporal range, spatio-temporal top-k nearest neighbor and spatio-temporal join query as case studies of how to exploit the spatio-temporal indexing in ST-Hadoop [6]. For the case of the spatio-temporal range query, ST-Hadoop exploits its *temporal hierarchy index* to select partitions that overlap with the temporal and spatial query predicates. An efficient algorithm that runs in three steps, temporal filtering, spatial search, and spatio-temporal refinement. (1) In the *temporal filtering* step, the hierarchy index is examined to select a subset of partitions that cover the temporal interval T. (2) Once the temporal partitions are selected, the *spatial search* step applies the spatial range query against each matched partition to select records that spatially match the query range A. (3) Finally, in the *spatio-temporal refinement* step, compares individual records returned by the spatial search step against the query interval T, to select the exact matching records. Similarly, there is a possibility that selected partitions might partially overlap with the query area A, and thus records outside the A need to be excluded from the final answer.

Distributed Processing (MapReduce and Dataflow). In the *MapReduce* layer, new implementations added inside SpatialHadoop MapReduce layer to enable ST-Hadoop exploits its spatio-temporal indexes and realizes spatio-temporal predicates. The implementation of MapReduce layer is based on MapReduce layer in SpatialHadoop [14], and just few changes were made to inject time awareness in this layer.

Query Language. The *Language* layer extends Pigeon language [12] to supports spatio-temporal data types (i.e., `STPOINT`, `TIME` and `INTERVAL`) and spatio-temporal operations (e.g., `OVERLAP`, `KNN` and `JOIN`) that take the advantages of the spatio-temporal index. Pigeon already equipped with several basic spatial predicates. ST-Hadoop changes the `OVERLAP` function to support spatio-temporal operations. ST-Hadoop extended the `JOIN` to take two spatio-temporal indexes as an input, and the processing of the *join* invokes the corresponding spatio-temporal procedure. ST-Hadoop extends `KNN` operation to finds top-k points to a given query point in space and time. ST-Hadoop computes the nearest neighbor proximity according to some α $(0 \leq \alpha \leq 1)$ value that indicates whether the *kNN* operation leans toward spatial, temporal, or spatio-temporal closeness. A ranking function computes the proximity between query point and any other points of the dataset.

Case-Studies of Applications. *Summit* [5] is a full-fledged open-source library on ST-Hadoop MapReduce framework with *built-in* native support for trajectory data. Summit cluster contains one master node that breaks a MapReduce job into smaller tasks, carried out by slave nodes. Summit modifies three core layers of ST-Hadoop, namely, *Language*, *Indexing* and *Operations*. The *Language* layer adds new SQL-Like interface for trajectory operations and data types. The modifications and the implementation of the *Indexing* (trajectory indexing) and *Operation* (trajectory range query, trajectory k nearest neighbor query and trajectory similarity query) layers are more complicated.

3.4 STARK

The **STARK** framework (https://github.com/dbis-ilm/stark) [21] is a promising new spatio-temporal data analytics framework (see in Fig. 9 its architecture). It is tightly integrated with Apache Spark [8] by leveraging Scala language features and it adds support for spatial and temporal data types and operations. Furthermore, STARK exploits SparkSQL functionality and implements SQL functions for filter, join with various predicates and aggregate vector as well as raster data. STARK also supports k nearest neighbor search and a density-based clustering operator allows to find groups of similar events. STARK includes spatial partitioning and indexing techniques for fast and efficient execution of the data analysis tasks.

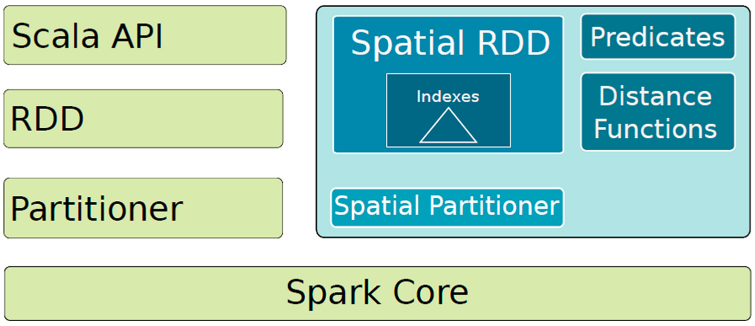


Fig. 9. STARK framework architecture [21].

Spatial and Spatio-Temporal Data Types. The main data structure of STARK is STObject. This class is a super-class of all spatial objects and provides a time component. STObject relies on the JTS library with the JTSplus extension, thus it supports all types of geometry objects, such as Point, Polygon, Linestring, Multipoint, Multypolygon and Multilinestring. Regarding the temporal data-type, the STObject contains o time component that facilitates temporal operations.

Spatial and Spatio-Temporal Storage (Indexing Techniques). The framework can index any partition, using an in memory spatial index structure. The R-tree index structure is currently supported by STARK, because of its JTS library dependency. Also, other indexing structures are planned to be included in future versions. There are three available indexing modes:

1. No Index: In some cases, indexing should be avoided (e.g. full table scan). No index mode should be used in these cases. When using no index mode, it does not matter how the RDD is partitioned.
2. Live Indexing: When live indexing is used, the framework firstly partitions the data items, in case they are not already partitioned. The spatio-temporal predicate is evaluated and the index is populated. Finally, the index is queried and the result is returned.
3. Persistent Index: The content of a partition is put into an index structure which can even be stored to disk and then used to evaluate the predicate. This execution mode transforms the input RDD from RDD[(STObject, V)] to RDD[RTree[STObject, (STObject, V)]]. After this transformation the resulting RDD consists of R-tree objects instead of single tuples. Multiple subsequent operations can benefit from these indexes. Furthermore, the same index can be used among different scripts, eliminating costly index creation time.

Spatial and Spatio-Temporal Partitioning Techniques. STARK is taking advantage of the Hadoop environment, resulting to parallel execution on cluster nodes. Every node processes a fragment of the whole dataset, which is call a partition. STARK spatial and spatio-temporal partitioning does not utilize Spark's built-in partitioners, for example a hash partitioner. Currently STARK uses only spatial partitioning, temporal partitioning is under development. In order to take advantage of the locality of data, STARK uses the following partitioners:

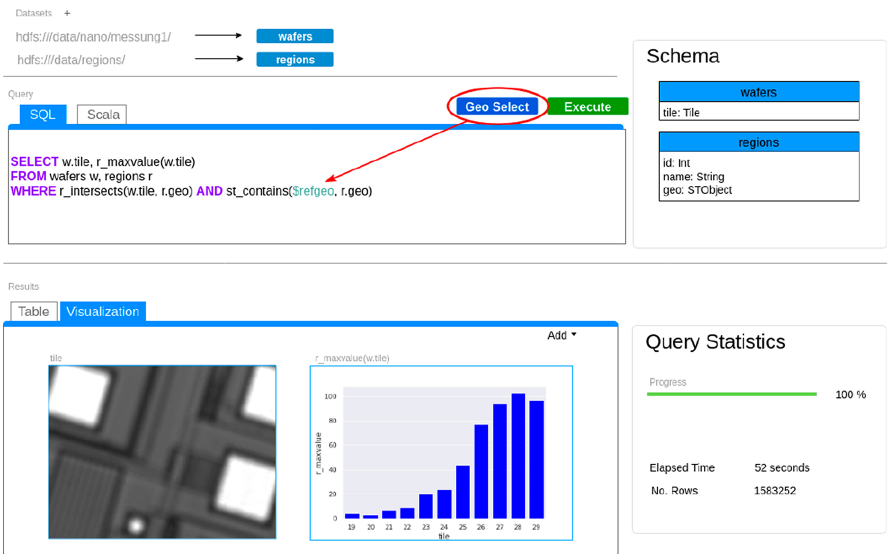
1. Grid Partitioner: The Grid Partitioner, evenly divides the dimensions based on a grid over the data space. The number of partitions per dimension are given as parameters. The disadvantage of grid partitioning is that spatial objects are not evenly distributed within the grid's partitions. As a result, some partitions are nearly empty, while other are contain most of the objects.
2. Binary Space Partitioner: Binary Space Partitioner (BSP) computes its partitions based on a maximum cost, which is given as a parameter. This is done by firstly dividing the data space into small quadratic shells, with a given side length. Then the partitioner evaluates all possible partitioning candidates along the cell bounds and then continues with testing all possible candidates. Finally, the partitioning with smallest cost difference between both candidate partitions is applied. The whole process results to two partitions and repeats itself recursively, if the according partition is longer than one cell length in at least one dimension and its cost is greater than the given maximum cost.
3. Partitioning Polygons: The spatial partitioners decide the preferred partition for each spatial object. In case the object is a point, the partitioner checks which cell contains the point and the assigns it to its relevant partition. When the spatial object is a polygon, even if this polygon is bigger than the partitions, the partitioner calculates its centroid point and then assign it the same way as if it was a point.

Spatial and Spatio-Temporal Operations. STARK supports most of the spatial and temporal operations. All operations rely on the STObject class and its spatial and temporal component. When the temporal component is missing, operations check only the spatial one. The STOBject class provides the following filter functions: intersect, contains and containedBy. Moreover, STARK implements the following operations: join, nearest neighbors, clustering and skyline (currently in development). All operations benefit from the underlying spatial and temporal partitioners and additionally from a partition-local spatial or temporal indexing.

Distributed Processing (MapReduce and Dataflow). STARK is fully integrated into Spark, so it benefits from Spark's DAG (Directed Acyclic Graph) execution model. The DAG scheduler transforms a logical execution plan to a physical execution plan.

Query Language. Besides the Scala API based on the core RDDs, STARK is integrated into SparkSQL and implements SQL functions to filter, join, and aggregate vector and raster data.

Visualization. STARK is heavily depended on Spark's capabilities; therefore the visualization tools of Spark can be used to visualize STARK data. There is only one documented visualization tool designed especially for STARK spatial visualization (see Fig. 10), which also combines raster data in final layout. This visualization tool comes with a web interface [22] where users can interactively explore raster and vector data using SQL.



```
SELECT w.tile, r_maxvalue(w.tile)
FROM wafers w, regions r
WHERE r_intersects(w.tile, r.geo) AND st_contains($refgeo, r.geo)
```

wafers
tile: Tile

regions
id: Int
name: String
geo: STObject

Elapsed Time	52 seconds
No. Rows	1583252

Fig. 10. STARK Visualization, Web Interface [22].

3.5 Comparison of Systems

In Table 1, we compare the four systems presented in the previous sections, regarding the features included in the presentation of these systems. Note that, there was non-available (N.A.) information available in the literature regarding some features of certain systems (language for GeoSpark, visualization for ST-hadoop and applications for GeoSpark and STARK).

4 More Big Spatial and Spatio-Temporal Data Analytics Systems

Apart from the previous four most representative data analytics systems that are actively maintained, we can find much more. They can be classified in four categories depending on whether are Hadoop-based or Spark-based [31,40], or spatial or spatio-temporal.

4.1 Hadoop-Based Big Spatial Data Analytics Systems

Hadoop-GIS [1] is scalable, high performance spatial data-ware housing system running on Hadoop. It utilizes SATO spatial partitioning (similar to *k*d-tree) and local spatial indexing to achieve efficient query processing. Hadoop-GIS uses global partition indexing to achieve efficient query results. Hadoop-GIS is supported with Hive, Pig and Scope. Hadoop-GIS supports fundamental spatial queries such as point, containment, join, and complex queries such as spatial cross-matching (large scale spatial join) and nearest neighbor queries. However, it lacks the support of complex geometry types including convex/concave polygons, line string, multi-point, multi-polygon, etc. HadoopGIS visualizer can plot images on the master node.

Parallel Secondo [28] integrates Hadoop with SECONDO, that is a database that can handle non-standard data types, i.e., spatial data. It employs Hadoop as the distributed task manager and performs operations on a multinode spatial DBMS. It supports the common spatial indexes and spatial queries except *kNN*. However, it only supports uniform spatial data partitioning techniques, which cannot handle the spatial data skewness problem. In addition, the visualization function needs to gather the data to the master node for plotting.

Esri GIS tools for Hadoop [18] are open source tools which would run on the ArcGIS platform. These allows integration of the Hadoop with Spatial data analytics software, i.e., ArcGIS Desktop. These tools work with big spatial data (big data with location) and allow you to complete spatial analysis using the power of distributed processing in Hadoop. For instance, (1) run a filter and aggregate operations on billions of spatial data records based on location; (2) define new areas (polygons) and run a point in polygon analysis on billions of spatial data records inside Hadoop; (3) visualize analysis results on a map and apply informative symbology; (4) integrate your maps in reports, or publish them as map applications online; etc.

Table 1. Overview of the comparative criteria of big spatial and spatio-temporal data analytics systems

	GeoSpark	SpatialHadoop	ST-Hadoop	STARK
Datatypes	Point, Rectangle, LineString, Polygon	Point, Rectangle, LineString, Polygon	STPoint, Time, Interval	Point, Polygon, Linestring, Multipoint, Multypolygon, Multilinestring, Time, Interval
Indexes	R-tree, Quadtree	R-tree	Temporal hierarchy index, Temporal Slicing, Spatial index	R-tree
Partitioning	Quadtree, k-d tree, STR-tree, Voronoi, Uniform, Hilbert	Quadtree, STR-tree, STR+, k-d tree, Hilbert-curve, Z-curve	Time-partitioning slicing, Data-partitioning slicing	Grid Partitioning, Binary Space Partitioning
Operations	Range, kNN, Spatial join, Distance join	Range, kNN, Spatial join	Spatio-temporal range, spatio-temporal top-k nearest neighbor, spatio-temporal join	Intersect, contains, containedBy, spatial join, nearest neighbors, clustering, skyline
Processing	DAG execution model	MapReduce	MapReduce	DAG execution model
Language	N.A	Pigeon	Pigeon	Piglet, Pig Latin
Visualization	GeoSparkViz	Single level image, Multilevel images	N.A	Web UI
Applications	N.A	MNTG, TAREEG, TAGHREED, SHAHED, HadoopViz	Summit	N.A

GeoWave [35] is a software library that connects the scalability of distributed computing frameworks and key-value stores with modern geospatial software to store, retrieve and analyze massive geospatial datasets. GeoWave indexes multidimensional data in a way that ensures values close together in multidimensional space are stored physically close together in the distributed datastore of your choice, by using Space Filling Curves (SFCs). GeoWave provides Hadoop input and output formats for distributed processing and analysis of geospatial data. GeoWave allows geospatial data in Accumulo platform to be shared and visualized via OGC standard services.

ScalaGiST [27] (Scalable Generalized Search Tree) is a scalable and non-intrusive indexing framework for Hadoop-MapReduce systems. It is based on classical Generalized Search Tree (GiST). ScalaGiST is designed for dynamic distributed environments to handle large-scale datasets and adapt to changes in the workload while leveraging commodity hardware. ScalaGiST is extensible in terms of both data and query type. It supports multiple types of indexes and can be dynamically deployed on large clusters while resilient to machine failures.

4.2 Spark-Based Big Spatial Data Analytics Systems

SIMBA [38] (Spatial In-Memory Big data Analytics) extends the Spark SQL engine to support spatial queries and analytics through SQL and the DataFrame API. Simba partitions data in a manner that they are of proper and balanced size and gathers records that locate close to the same partition (STR is used by default). It builds a local index per partition and a global index by aggregating information from local indexes. Simba builds local R-tree indexes on each DataFrame partition and uses R-tree grids to perform the spatial partitioning. It supports range and kNN queries, kNN and distance joins.

SpatialSpark [39] is a lightweight implementation of spatial support in Apache Spark. It targets in-memory processing for higher performance. SpatialSpark supports several spatial data types including points, linestrings, polylines, rectangles and polygons. It supports three spatial partitioning schemes fixed Grid, binary split and STR partitioning. The indexing is supported used R-trees. SpatialSpark offers range queries and spatial joins between various geometric objects.

LocationSpark [34] is an ambitious project, built as a library on top of Spark. It requires no modifications to Spark and provides spatial query APIs on top of the standard operators. It supports a wide range of spatial features. It provides Dynamic Spatial Query Execution and operations (Range, kNN, Insert, Delete, Update, Spatial-Join, kNN-Join, Spatio-Textual). The system builds two indexes, a global (Grid, Quadtree and a Spatial-Bloom Filter) and a local per-worker, user-decided index (Grid, R-tree, Quadtree and IR-tree). Global index is constructed by sampling the data. Spatial indexes are aiming to tackle unbalanced data partitioning. Additionally, the system contains a query scheduler, aiming to tackle query skew.

Magellan [32] is a distributed execution engine for spatial analytics on big data. It leverages modern database techniques in Apache Spark like efficient data layout, code generation, and query optimization in order to optimize spatial queries. Magellan extends SparkSQL to accommodate spatial datatypes, geometric predicates and queries. Magellan supports several spatial data types like points, linestrings, rectangles, polygons, multipoints and multipolygons. It allows the user to build a Z-curve index on spatial objects. Magellan supports range queries and spatial joins.

SparkGIS [9] a distributed, in-memory spatial data processing framework to query, retrieve, and compare large volumes of analytical image result data for algorithm evaluation. SparkGis combines the in-memory distributed processing capabilities of Apache Spark and the efficient spatial query processing of Hadoop-GIS. SparkGIS mitigates skew by making available various partitioning schemes as previously evaluated on MapReduce. SparkGIS uniquely improves memory management in spatial-processing Spark jobs by spatially aware management of partitions loaded into memory rather than arbitrary spilling to disk. The performance of SparkGIS was proved with medical pathology images and with OpenStreetMap (OSM) data.

GeoTrellis [26] is an open source, geographic data processing library designed to work with large geospatial raster data sets. GeoTrellis leverages Apache Spark for distributed processing. GeoTrellis relies on the data being exposed using an HDFS filesystem with the individual files written using the GeoTIFF format. Distributed processing relies on indexing large datasets based on a multi-dimensional space-filling curve (SFC), since SFCs enable the translation of multi-dimensional indices into a single-dimensional one, while maintaining geospatial locality. GeoTrellis includes some operations using vector and point data to support raster data operations.

Other big spatial data analytics systems are GeoMatch and SciSpark. **GeoMatch** [43] is a scalable and efficient big-data pipeline for large-scale map matching on Apache Spark. GeoMatch utilizes a novel spatial partitioning scheme inspired by Hilbert SFC, generating an effective indexing technique based such SFC that expedites spatial query processing in a distributed computing environment. Once the index has been built, GeoMatch uses an efficient and intuitive load balancing scheme to evenly distribute the parts of the index between available computing cores. **SciSpark** [37] is a big data framework that extends Apache Spark's in-memory parallel computing to scale scientific computations. The current architecture of SciSpark includes: (1) time and space partitioning of high resolution geo-grids from NetCDF3/4; (2) a sciDataset class providing n-dimensional array operations; (3) parallel computation of time-series statistical metrics; and (4) an interactive front-end using science (code and visualization) Notebooks.

4.3 Hadoop-Based Big Spatio-Temporal Data Analytics Systems

CloST [33] is a Hadoop-based storage system for big spatio-temporal data analytics, based on MapReduce framework. CloST is targeted at fast data loading, scalable spatio-temporal range query processing and efficient storage usage to handle very large historical spatio-temporal datasets. A simple data model which has special treatments on three core attributes including an object id, a location and a time. Based on this data model, CloST hierarchically partitions data using all core attributes which enables efficient parallel processing of spatio-temporal range scans and efficient parallel processing of two simple types of spatio-temporal queries: single-object queries and all-object queries. CloST supports a parallel implementation of the R-tree index.

4.4 Spark-Based Big Spatio-Temporal Data Analytics Systems

BinJoin [36] is a Spark-based implementation of a spatio-temporal attribute join that runs in a distributed manner across a Hadoop cluster. One important conclusion obtained from the experimental study is that the most effective and efficient distributed spatial join algorithm depends on the characteristics of the two input datasets. For the implementation of the join algorithms was used a local index and a query optimizer. Finally, an interesting observation extracted from the experiments is that spatio-temporal near join was able to scale to larger input sizes than space-only near join, because the temporal condition alleviates the effects of spatial skew.

GeoMesa [23] is an open-source spatio-temporal index extension built on top of distributed data storage systems. It provides a module called GeoMesaSpark to allow Spark to read the preprocessed and preindexed data from Accumulo data store. GeoMesa also provides RDD API, DataFrame API and Spatial SQL API so that the user can run spatio-temporal queries on Apache Spark [24]. GeoMesa uses R-tree spatial partitioning technique to decrease the computation overhead. However, it uses a Grid file as the local index per DataFrame partition. GeoMesa supports range query and spatial join query.

5 Conclusions

In a world where the volume of available data is continuously expanding and, in numerous cases, the related data objects contain spatial and/or spatio-temporal characteristics, scalable (and, therefore, distributed) systems capable of modeling, storing, querying and analyzing big spatial and spatio-temporal data are a necessity for modern and emerging applications. This fact is verified by the large number of parallel and distributed systems for big spatial and spatio-temporal data management and analysis that have been developed. In this paper, we presented four selected such systems, considering their acceptance in the research and advanced applications communities. This presentation was structured along specific categories of key system features and intends to provide to the reader a view of the differences and similarities of these systems. These four systems are actively being maintained and updated. However, many more systems have been developed. To assist the reader to develop a more complete point of view of the large world of ecosystems supporting management of big spatial and spatio-temporal data, we also present in brief a number of them. This paper intends to provide information that would allow a researcher or practitioner choose a system that is most suitable for his/her needs or compare another system (that will be developed in the future, or is not covered in this paper) with the ones presented here.

References

1. Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., Saltz, J.H.: Hadoop-GIS: a high performance spatial data warehousing system over MapReduce. PVLDB **6**(11), 1009–1020 (2013)

2. Alarabi, L., Eldawy, A., Alghamdi, R., Mokbel, M.F.: TAREEG: a MapReduce-based web service for extracting spatial data from OpenStreetMap. In: SIGMOD Conference, pp. 897–900 (2014)
3. Alarabi, L., Mokbel, M.F.: A demonstration of ST-hadoop: a MapReduce framework for big spatio-temporal data. PVLDB **10**(12), 1961–1964 (2017)
4. Alarabi, L., Mokbel, M.F., Musleh, M.: ST-Hadoop: a MapReduce framework for spatio-temporal data. In: SSTD Conference, pp. 84–104 (2017)
5. Alarabi, L.: Summit: a scalable system for massive trajectory data management. SIGSPATIAL Special **10**(3), 2–3 (2018)
6. Alarabi, L., Mokbel, M.F., Musleh, M.: ST-Hadoop: a MapReduce framework for spatio-temporal data. GeoInformatica **22**(4), 785–813 (2018). https://doi.org/10.1007/s10707-018-0325-6
7. Apache. Hadoop. http://hadoop.apache.org/
8. Apache. Spark. http://spark.apache.org/
9. Baig, F., Vo, H., Kurç, T.M., Saltz, J.H., Wang, F.: SparkGIS: resource aware efficient in-memory spatial query processing. In: SIGSPATIAL/GIS Conference, pp. 28:1–28:10 (2017)
10. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: OSDI Conference, pp. 137–150 (2004)
11. Eldawy, A., Li, Y., Mokbel, M.F., Janardan, R.: CG_Hadoop: computational geometry in MapReduce. In: SIGSPATIAL/GIS Conference, pp. 284–293 (2013)
12. Eldawy, A., Mokbel, M.F.: Pigeon: a spatial MapReduce language. In: ICDE Conference, pp. 1242–1245 (2014)
13. Eldawy, A., Mokbel, M.F.: The ecosystem of SpatialHadoop. SIGSPATIAL Special **6**(3), 3–10 (2014)
14. Eldawy, A., Mokbel, M.F.: SpatialHadoop: a MapReduce framework for spatial data. In: ICDE Conference, pp. 1352–1363 (2015)
15. Eldawy, A., Alarabi, L., Mokbel, M.F.: Spatial partitioning techniques in spatial hadoop. PVLDB **8**(12), 1602–1605 (2015)
16. Eldawy, A., Mokbel, M.F., Al-Harthi, S., Alzaidy, A., Tarek, K., Ghani, S.: SHAHED: a MapReduce-based system for querying and visualizing spatio-temporal satellite data. In: ICDE Conference, pp. 1585–1596 (2015)
17. Eldawy, A., Mokbel, M.F., Jonathan, C.: HadoopViz: a MapReduce framework for extensible visualization of big spatial data. In: ICDE Conference, pp. 601–612 (2016)
18. ESRI-GIS: GIS Tools for Hadoop (2014). http://esri.github.io/gis-tools-for-hadoop/. Accessed 20 July 2019
19. Garcia-Garcia, F., Corral, A., Iribarne, L., Mavrommatis, G., Vassilakopoulos, M.: A comparison of distributed spatial data management systems for processing distance join queries. In: ADBIS Conference, pp. 214–228 (2017)
20. García-García, F., Corral, A., Iribarne, L., Vassilakopoulos, M., Manolopoulos, Y.: Efficient large-scale distance-based join queries in spatialhadoop. GeoInformatica **22**(2), 171–209 (2017). https://doi.org/10.1007/s10707-017-0309-y
21. Hagedorn, S., Goetze, P., Sattler, K.U.: he STARK framework for spatio-temporal data analytics on spark. In: BTW Conference, pp. 123–142 (2017)
22. Hagedorn, S., Birli, O., Sattler, K.U.: Processing large raster and vector data in apache spark. In: BTW Conference, pp. 551–554 (2019)
23. Hughes, N.J., Annex, A., Eichelberger, C.N., Fox, A., Hulbert, A., Ronquest, M.: Geomesa: a distributed architecture for spatio-temporal fusion. In: Geospatial Informatics, Fusion, and Motion Video Analytics V, vol. 9473, p. 94730F. International Society for Optics and Photonics (2015)

24. Hulbert, A., Kunicki, T., Hughes, J.N., Fox, A.D., Eichelberger, C.N.: An experimental study of big spatial data systems. In: BigData Conference, pp. 2664–2671 (2016)
25. Jiang, D., Ooi, B.C., Shi, L., Wu, S.: The performance of MapReduce: an in-depth study. PVLDB **3**(1), 472–483 (2010)
26. Kini, A., Emanuele, R.: Geotrellis: adding geospatial capabilities to spark. Spark Summit (2014)
27. Lu, P., Chen, G., Ooi, B.C., Vo, H.T., Wu, S.: ScalaGiST: scalable generalized search trees for MapReduce systems. PVLDB **7**(14), 1797–1808 (2014)
28. Lu, J., Güting, R.H.: Parallel secondo: boosting database engines with hadoop. In: ICPADS Conference, pp. 738–743 (2012)
29. Magdy, A., Alarabi, L., Al-Harthi, S., Musleh, M., Ghanem, T.M., Ghani, S., Mokbel, M.F.: Taghreed: a system for querying, analyzing, and visualizing geotagged microblogs. In: SIGSPATIAL/GIS Conference, pp. 163–172 (2014)
30. Mokbel, M.F., Alarabi, L., Bao, J., Eldawy, A., Magdy, A., Sarwat, M., Waytas, E., Yackel, S.: MNTG: an extensible web-based traffic generator. In: SSTD Conference, pp. 38–55 (2013)
31. Pandey, V., Kipf, A., Neumann, T., Kemper, A.: How good are modern spatial analytics systems? PVLDB **11**(11), 1661–1673 (2018)
32. Sriharsha, R.: Magellan: Geospatial Analytics Using Spark (2015). https://github.com/harsha2010/magellan. Accessed 20 July 2019
33. Tan, H., Luo, W., Ni, L.M.: CloST: a hadoop-based storage system for big spatio-temporal data analytics. In: CIKM Conference, pp. 2139–2143 (2012)
34. Tang, M., Yu, Y., Malluhi, Q.M., Ouzzani, M., Aref, W.G.: LocationSpark: a distributed in-memory data management system for big spatial data. PVLDB **9**(13), 1565–1568 (2016)
35. Whitby, M.A., Fecher, R., Bennight, C.: GeoWave: utilizing distributed key-value stores for multidimensional data. In: SSTD Conference, pp. 105–122 (2017)
36. Whitman, R.T., Park, M.B., Marsh, B.G., Hoel, E.G.: Spatio-temporal join on apache spark. In: SIGSPATIAL/GIS Conference, pp. 20:1–20:10 (2017)
37. Wilson, B., Palamuttam, R., Whitehall, K., Mattmann, C., Goodman, A., Boustani, M., Shah, S., Zimdars, P., Ramirez, P.M.: SciSpark: highly interactive in-memory science data analytics. In: BigData Conference, pp. 2964–2973 (2016)
38. Xie, D., Li, F., Yao, B., Li, G., Zhou, L., Guo, M.: Simba: efficient in-memory spatial analytics. In: SIGMOD Conference, pp. 1071–1085 (2016)
39. You, S., Zhang, J., Gruenwald, L.: Large-scale spatial join query processing in cloud. In: ICDE Workshops, pp. 34–41 (2015)
40. Yu, J., Sarwat, M.: Geospatial data management in apache spark: a tutorial. In: ICDE Conference, pp. 2060–2063 (2019)
41. Yu, J., Zhang, Z., Sarwat, M.: GeoSparkViz: a scalable geospatial data visualization framework in the apache spark ecosystem. In: SSDBM Conference, pp. 15:1–15:12 (2018)
42. Yu, J., Zhang, Z., Sarwat, M.: Spatial data management in apache spark: the GeoSpark perspective and beyond. GeoInformatica **23**(1), 37–78 (2018). https://doi.org/10.1007/s10707-018-0330-9
43. Zeidan, A., Lagerspetz, E., Zhao, K., Nurmi, P., Tarkoma, S., Vo, H.T.: GeoMatch: efficient large-scale map matching on apache spark. In: BigData Conference, pp. 384–391 (2018)