Spatial Data Processing
A Framework Survey

Ayman Zeidan
Department of Computer Science
CUNY Graduate Center
365 5th Ave
New York, NY, USA

Professor Huy T. Vo, Advisor
Outline

- Introduction
- Challenges
- Need for Spatial Frameworks
- General Approach
- Hadoop-based Frameworks
  - Esri GIS Tools for Hadoop
  - Hadoop-GIS
  - SpatialHadoop
- Spark-based Frameworks
  - SpatialSpark
  - GeoSpark
  - LocationSpark
  - STARK
  - Simba
- Different experiments, different results
- Ideally
Introduction

- Information Explosion
  - Data is arriving faster than we can process/analyze
  - Logs, tracking, usages, IoT ...
  - But we can store it!
  - Big data
  - 3Vs (4Vs)
    - Spatial component

- Lots of examples
  - Wikibon: big data market will reach over $84Bil by 2026\(^1\)
  - Facebook warehouse stores 300PB with 600TB daily incoming\(^2\)
  - Boeing 787 Dreamliner: can generate up to 500GB per flight\(^3\)

- Same goal:
  - Unlock hidden values to improve and invent
Introduction – Continued

- Processing challenges
  - Single Machine vs Parallel
  - Speed, real-time, cost, scaling up vs out

- Big Data processing Frameworks
  - A form of cluster computing
  - In parallel rather than sequential
  - Not spatially aware

- Hadoop and Spark
  - Most popular (commodity hardware, easy to use, maintain, scale...)
  - Common frameworks for developing distributed programs
  - Abstract complex operations, low-level data communication, concurrency controls
  - Generic data processing
  - Cannot recognize Spatial data and operations
  - Specialized frameworks that tap into their ecosystem
Challenges in Spatial Data Processing

- Spatial data is different
  - Need native support
- Mixed with non-spatial
  - How much of it will make to the next step
  - Effects performance (RAM, Disk)
- Many shapes and sizes
  - Point, Polygon, LineSegment, LineString ...
  - How many can be supported – MBR as a substitute
  - Open Geospatial Consortium
- Data at Rest
  - Structure/Semi/Un
  - Frameworks should not restrict structure
  - Homo/heterogeneous datasets, uniform or not
Challenges in Spatial Data Processing – continued

- **Operations**
  - Range: set of all overlapping records
  - Contains: true/false
  - Join: pair of elements that satisfy one of *many* predicates (distance, overlap...)
  - $k$NN: $k$NN join, $k$NN distance
  - Most systems surveyed implement over Point/MBR and Point/Polygons
  - How many can be supported?

- **Usability and integration**
  - Usability not an interesting research, but necessary for usable systems (non CS)
  - Goal to make a framework that can be extended with new objects and operations
  - Integration done using SQL, Hive, Pig Latin, RDD, DataFrame
  - Integration into other tasks via intermediate HDFS writes or RDD transformations

Spatial Data Processing Frameworks
Challenges in Spatial Data Processing – continued

- **Indexing**
  - Speeds up spatial operation
  - Spatial data employ 2-level indexing (Global and Local)
  - Grid, R-Tree, R*-Tree, BSP, Quadtree, K-D Tree, SFC

- **Temporal**
  - Takes timestamp into consideration

- **Interactive and batch processing**
  - For different sizes: small/large, large/large

- **Scalability**
  - How much can be handle while adding resources
  - How new resources are utilized

- **Reliability**
  - Hardware, HDFS, RDD (lineage graph)
Need for Spatial Frameworks

- **Apache Hadoop and Spark**
  - Single machines inefficient
  - Scales up so much before scaling out
  - Most popular
  - Abstract complex and error-prone procedures, low-level data com., and conc. controls

- **Hadoop and Spark are ill-equipped for spatial data**
  - Treat all data the same
  - Don’t recognize special shapes
  - Can be used, but inefficient

- **Spatial frameworks make Hadoop and Spark spatially aware**
  - Same from a user’s view (input → query → output)
  - Differ in algorithm, integration, language, objects, and operations
General approach

Almost all frameworks (Hadoop or Spark) follow the same flow:

1. Read datasets and build objects
2. Sample one or both
3. Build grid (master or distributed then master) using MBR
4. Repartition data (shuffling)
5. Account for partition skews
6. Apply spatial operation using MBR
7. Refine results using a spatial library (optional in some)
Hadoop-based Frameworks

- Hadoop
  - HDFS + MR
  - 2003: Google’s GFS and MR
  - 2006: Yahoo’s HDFS and MR \(\rightarrow\) Apache

- A number of attempts to make Hadoop spatially aware
  - Esri GIS Tools for Hadoop, Hadoop-GIS, SpatialHadoop

- Techniques focus on:
  - Making Hadoop spatially aware
  - Ease of use
  - Speed
  - Scalability
Esri GIS Tools for Hadoop

- Developed for use with ArcGIS
  - Mainly for geometry filtering
  - Works with GB sized datasets
  - Aims to be OGC compliant

- 3 layers on top of Hadoop
  - Esri Geometry API for Java (sp. Objects, operations, indexing) – Java
  - Written as Hive UDFs – Java
  - Geoprocessing Tools for Hadoop (ArcGIS connection tools) – Python

- $k$NN join (no global index)
  - HQL → MR → Index Polygon Dataset (Local Partitions) → Points checked against all Polygons (shuffle) → Aggregate Results → Write to HDFS (JSON)

- Global index used for aggregate hotspot query

- Drawbacks: IO intensive, for ArcGIS, data sample, poor indexing, RAM usage

Spatial Data Processing Frameworks
Hadoop-GIS

- 3 layers on top of Hadoop
  - Query Language (Spatial support via Hive UDFs)
  - Query Translation (HQL → MR)
  - Real-time Spatial Query Engine (RESQUE) (indexing & querying)

- Streaming approach

- Mix of Java, C++, Python (LibSpatialIndex, GEOS)

- Global spatial indexing (Grid) and optional local indexing (Grid/R-Tree)

- Join Query
  - HQL → Sample datasets → Partition MBRs → HDFS → Build Global Index → Assign buckets → Global Partitioning (Union) → Local Index → Query → Sort → Remove duplicates

- Partition skew lightly handled through grid cell splits

- Drawbacks: installations\(^1\), IO, sampling, non-spatial data, poor indexing, sorting

Spatial Data Processing Frameworks

11
SpatialHadoop

- Uses Hadoop APIs for tighter integration
- 4 layers into Hadoop
  - Storage (file indexing)
  - MapReduce (spatially aware MR jobs)
  - Operations (spatial support)
  - Language (Pigeon (Pig-Latin))
- Uses configuration files
- Limits samples and indexes to 64MB (single block when written to HDFS)
- Join Query
  - Pigeon → Sample datasets → Local Index → HDFS → Global Index (Merge indexes) → read data (MBR) → partition → shuffle → Local Index → Query
- Duplicates: intersect results and query area
- Drawbacks: configuration, IO, sampling, need to learn Pigeon, duplicate removal
Hadoop FWs comparison

- Development of Hadoop-GIS seems to have stopped
- Neither one is easily extended to include new objects and operations
- Esri GIS Tools for Hadoop is mainly for ArcGIS
- Hadoop-GIS major weakness:
  - Streaming, GEOS
- SpatialHadoop is faster than Hadoop-GIS
  - Less HDFS read/write
  - No streaming
  - More spatially aware
- Hadoop-GIS global index is slightly smaller than SpatialHadoop
- SpatialHadoop can handle larger datasets than Hadoop-GIS
<table>
<thead>
<tr>
<th>Feature</th>
<th>Esri GIS Tools</th>
<th>Hadoop-GIS</th>
<th>SpatialHadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release Date</td>
<td>2013</td>
<td>2013</td>
<td>2014</td>
</tr>
<tr>
<td>Last Update</td>
<td>2018</td>
<td>2012</td>
<td>2018</td>
</tr>
<tr>
<td>Integration Approach</td>
<td>On Top of Hadoop</td>
<td>On Top of Hadoop</td>
<td>Into Hadoop</td>
</tr>
<tr>
<td>Language Integration</td>
<td>HiveQL</td>
<td>HiveQL</td>
<td>Pigeon (Pig Latin)</td>
</tr>
<tr>
<td>OGC Compliant</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Geometry Library</td>
<td>Esri Geometry API Java</td>
<td>GEOS</td>
<td>JTS</td>
</tr>
<tr>
<td>Global Indexing</td>
<td>Grid (Partial)</td>
<td>Grid</td>
<td>Grid, R-Tree</td>
</tr>
<tr>
<td>Local Indexing</td>
<td>PMR Quadtree</td>
<td>R-Tree</td>
<td>R-Tree or R+-Tree</td>
</tr>
<tr>
<td>Index Persistence</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Pruning</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Carry non-spatial data</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Skew Handling Level</td>
<td>Partition</td>
<td>Partition</td>
<td>Partition</td>
</tr>
<tr>
<td>Mixed object Query</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Base Code</td>
<td>Java, Python (non-spatial tools)</td>
<td>Java, Python, and C++</td>
<td>Java</td>
</tr>
<tr>
<td>Allows user-defined operations</td>
<td>No, (base-code modifications required)</td>
<td>No, (base-code modifications required)</td>
<td>No, (base-code modifications required)</td>
</tr>
<tr>
<td>Installation</td>
<td>None</td>
<td>GEOS, libspatialindex</td>
<td>None</td>
</tr>
<tr>
<td>Configuration</td>
<td>No</td>
<td>System environment variables and Configuration Files</td>
<td>Configuration Files</td>
</tr>
<tr>
<td>Required Expertise Level</td>
<td>Regular Hive</td>
<td>Regular Hadoop</td>
<td>Advanced Hadoop</td>
</tr>
<tr>
<td>Spatial Objects</td>
<td>Point, Polygon, Line, Envelope (MBR)</td>
<td>Point, Box, Polygon</td>
<td>Point, Circle, Rectangle, LineString</td>
</tr>
<tr>
<td>Spatial Operations</td>
<td>Range, kNN, Join</td>
<td>Range, kNN, Join</td>
<td>Range, kNN, Join</td>
</tr>
</tbody>
</table>
Spark-based Frameworks

- Spark
  - 2010: UC Berkeley's AMPLab releases open source Spark
  - 2013: Apache Spark
  - Backward compatible with Hadoop
  - Solved 2 major Hadoop drawbacks
    - In-memory processing (RDDs)
    - Disk fault tolerance (Lineage Graph)

- A number of attempts to make Hadoop spatially aware
  - SpatialSpark, GeoSpark, LocationSpark, STARK, Simba

- Techniques focus on:
  - Ease of use
  - Expandability
  - Speed
  - Scalability
SpatialSpark

- Relies on Spark’s RDDs
- Used as a library (Scala based)
- Doesn’t build any special layers on top of Spark
- Join Query
  - RDD → sample one set → partitions’ MBRs → grid index → broadcast index → repartition
    
    \[(groupBy \rightarrow join)\] (object duplicate) → local index → Query
  
- Partition skew not taken into consideration
- No consideration for duplicate results
- Loads all data into memory (dataset \(\leq\) RAM)
- Drawbacks: RAM \((groupBy)\), broadcast, data shuffle, sampling, limited to joins, no skew mitigation.
GeoSpark

- 2 Layers on top of Spark
  - Spark layer remains as is
  - SRDD (spatial objects and operations)
  - Query Processing (spatial queries)

- Used as a library by extending RDDs

- Can create SRDDs from supported format (strict format on unstructured data)

- Join Query
  - SRDD → Sample both datasets → Grid (even counts) → repartition (object duplicate) → local index (cost-based) → Query

- Range query uses broadcast of query window

- Duplicates resolved via group and sort operations

- Drawbacks: RAM, MBR Results, multiple passes over datasets, sampling, no query skew mitigation, no control over local index (except type)
LocationSpark

- 2 Layers on top of Spark
  - Spark layer remains as is
  - Query Scheduler (Data partitioning)
  - Query Executor

- Used as a library by extending RDDs

- Targets query skews via a spatial bloom filter

- Join Query
  - SRDD → Sample both datasets → Grid (even counts) → Repartition (object duplicate), sFilter\(^1,2\) → replicate inner table to outer → local index → Query

- Reduce memory: serialize less used objects to disk

- Results are aggregated, objects discarded

- Drawbacks: output, RAM, relies on disk, table broadcast, multiple passes over set, sampling, limited objects (Point, Box)
STARK

- Spatio-temporal framework
  - Limits temporal to some operations (no indexing)

- 4 Layers on top of Spark
  - Spatial RDD (spatio-temporal objects)
  - Predicates (contains, distance, intersects)
  - Distance functions (functions for Cartesian and geodetic distances)
  - Language (Piglet – Pig Latin spatial constructs)

- Used as a library

- Join Query
  - RDD → compute datasets’ MBR → Grid or BSP → repartition (duplicate) → Query

- User interaction required
  - Global grid type (Grid or BSP)
  - Use local index or not
Polygons are assigned by centroid.

- Oversized polygons are accounted for by computing the partition’s extent (virtual)
- If partitions grow they are split to balance them out

Drawbacks: Partition-level skew, loss of generality, works best if user knows their datasets (index or not).
Simba

- Target multidimensional objects (R-Tree)
  - No support for time

- 4 Layers on top of Spark
  - SQL Parser layer (spatial query parser)
  - Spatial (Objects and Operations)
  - Query Optimizer (CBO for plans)
  - Index Manager (indexing support)

- Used as a library

- Works from DataFrame or Spark SQL
  - Benefits from Spark’s multi-threading module

- Datasets are treated as tables and records are stored as row objects.
  - Table is loaded as a RDD<Row> objects.
  - Point/Polygon Queries only (independent implementation) – kNN, Join, kNN Join
Simba – Continued

- **Indexing:**
  - Packs rows into an array which speeds up sampling (RAM and time overhead, but Simba claims it’s low)
  - Three steps: partition (skew aware) → local index (stats, partition MBR, # records) → back to Master → global index (Master’s memory)

- **Join Query**
  - Relation → Logical Plan → Logical Optimizer\(^1\) → Physical Plans\(^2\) → CBO\(^3\) → Best Physical Plan → Partition’s MBR & Count → Master → Repartition → Query

- **Not clear:**
  - Was index timing included in the runtimes since it is done against table/columns (not live)
  - Separately Simba beats all (index results then query)

- **Drawbacks:** Datasets as tables, Points over MBR only, RAM, shuffle\(^4\), partition-level skew, index re-compute with data change.
Spark FWs comparison

- Development of all frameworks is still active

- Similar from a high-level view
  - Rely mainly on in-memory processing then HDFS
  - Minimum support Point, and Polygon
  - Range, Join, $k$NN
  - Extendable to include new objects and operations (except for SpatialSpark)

- SpatialSpark and GeoSpark problems
  - Failing due to out of memory
  - GeoSpark was reported to be unable to perform combine partitioning and local indexing
  - SpatialSpark is not allowed for range queries.
Spark FWs comparison – continued

- **STARK:**
  - Better when indexing is used
  - Better runtime with live index (contains, intersects, withinD). GeoSpark then SpatialSpark
  - GeoSpark doesn’t support withinD
  - SpatialSpark doesn’t support intersects

- **Simba:**
  - Uses Spark SQL thread pooling which gives it an advantage
  - Indexing Time: GeoSpark, Simba, SpatialSpark
  - Indexing RAM: SpatialSpark, Simba, GeoSpark
  - Throughput, Latency: Simba, SpatialSpark, Geospark

- **Simba multidimensional:**
  - Time and memory linearly increase as dimension increases (2 – 6)
  - Throughput (Latency): Decrease(Increase) as dimension increases (2 – 6)
<table>
<thead>
<tr>
<th>Feature</th>
<th>SpatialSpark</th>
<th>GeoSpark</th>
<th>LocationSpark</th>
<th>STARK</th>
<th>Simba</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Update</td>
<td>2017</td>
<td>2018</td>
<td>2017</td>
<td>2018</td>
<td>2018</td>
</tr>
<tr>
<td>Integration Approach</td>
<td>On Top of Spark</td>
<td>On Top of Spark</td>
<td>Into Spark</td>
<td>Into Spark</td>
<td>Into Spark</td>
</tr>
<tr>
<td>Language Integration</td>
<td>-</td>
<td>-</td>
<td>Scala (Spatial RDD)</td>
<td>Figlet and Scala via RDD integration</td>
<td>DataFrame and Spark SQL</td>
</tr>
<tr>
<td>OGC Compliant</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Geometry Library</td>
<td>JTS</td>
<td>JTSPlus</td>
<td>JTS</td>
<td>JTS</td>
<td>Built-in</td>
</tr>
<tr>
<td>Global Indexing</td>
<td>Grid, K-D Tree</td>
<td>Grid</td>
<td>Grid, Quad-Tree</td>
<td>Grid, Cost-Based Binary Space, R-Tree</td>
<td>Grid, R-Tree, KD-tree</td>
</tr>
<tr>
<td>Local Indexing Options</td>
<td>None, R-Tree</td>
<td>None, R-Tree, Quad-Tree</td>
<td>None, Grid, R-tree, Quad-Tree, IR-tree</td>
<td>None, R-Tree</td>
<td>HashMaps, TreeMaps, R-Tree</td>
</tr>
<tr>
<td>Index Persistence</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Pruning</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Carry non-spatial data</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Slew Handling Level</td>
<td>-</td>
<td>Partition</td>
<td>Query</td>
<td>Partition</td>
<td>Partition</td>
</tr>
<tr>
<td>Mixed object Query</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Base Code</td>
<td>Scala</td>
<td>Java</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
</tr>
<tr>
<td>Allow for new operations</td>
<td>No</td>
<td>Yes (SRDD and query processing layers)</td>
<td>Yes (Spatial Objects, Query Scheduler, Query Executor)</td>
<td>Yes (Spatial-RDDFunctions and Figlet)</td>
<td>Yes (Requires source code modification)</td>
</tr>
<tr>
<td>Installation</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Configuration</td>
<td>In code</td>
<td>In code</td>
<td>In code</td>
<td>In code</td>
<td>In code</td>
</tr>
<tr>
<td>Required Expertise Level Spark</td>
<td>Moderate Spark</td>
<td>Regular Spark</td>
<td>Regular Spark</td>
<td>Regular Spark</td>
<td>Regular Spark DataFrame</td>
</tr>
<tr>
<td>Spatial Objects</td>
<td>MBR</td>
<td>Circle, LineString, Point, Polygon, and Rectangle</td>
<td>Box and Point</td>
<td>Inherited from JTS</td>
<td>Circle, MBR, Point, Polygon</td>
</tr>
<tr>
<td>Spatial Operations</td>
<td>Range, Join, kNN</td>
<td>Range, Join, kNN</td>
<td>Range, kNN, Join, kNN-join</td>
<td>Filtering, Join, kNN, Clustering</td>
<td>Range, Join, kNN</td>
</tr>
</tbody>
</table>

25
Different experiments, different results

- Need a uniform way to evaluate frameworks
  - A common set of rules for fair evaluations
  - Same test environment (Version: HDFS, Spark, Java, Scala)
  - Similar datasets. Maybe synthetic ones (varying density)
  - Focus on same features (operations, objects, extending)

- Test:
  - Time and memory required to build local and global indexes
  - Space required for the indexes

- Scalability:
  - Results for varying the input and/or processing power

- Results:
  - Accuracy and completeness
  - Compare to those achieved via naive approach
Ideally

- Multidimensional support
  - Spatio-temporal support at minimum

- Objects
  - Basics: not just MBR, but Point, Polygon, LineString ...
  - Allow for new objects to be added

- Operations
  - Basic: Join, kNN, Range
  - Predicates: distance, within, equals, includes
  - Mixed object
  - Allow for new operations to be added
  - Batch vs live search

- Persisted and live Indexing
  - Basics: R-Tree, Grid, QuadTrees
  - Allow for new techniques to be added
Ideally – continued

- **Partitioning**
  - Fastest way with lowest shuffling
  - Works for all objects and operations
  - Hetero/homogenous datasets

- **Scalability**
  - Terabytes, petabytes ...
  - Make efficient use of new resources

- **Language integration**
  - Scala/Java/Python/R
  - RDD/DataFrame
  - SQL