GEOSPATIAL DATA MANAGEMENT IN APACHE SPARK

Presented by:
Jia Yu
Mohamed Sarwat

DatSys
@ASU
About Us

• Data Systems Lab: a research lab at Arizona State University
• Geospatial databases, distributed geospatial data management
• GeoSpark: open-sourced in April 2015
GeoSpark

http://datasystemslab.github.io/GeoSpark/

- Spatial RDD, SQL, DataFrame on Apache Spark
- Distributed map visualization (Viz SQL)
- Integrated with Apache Zeppelin
- Built-in large-scale traffic simulator

Google “GeoSpark ASU”

In production!

8K - 10K monthly downloads
Outline

- Big Geospatial Data
- Manage Spatial Data
- Manage Spatio-Temporal Data
- Spatial Data Analytics in Spark
Geospatial Data

• Mobile devices - 4.68 billion in 2019
Geospatial Data

• IoT sensors in Smart City: 7 billion in 2019

Massive LiDar data

1.5 billion taxi trips
Geospatial Data

• Climate monitoring: 22 PB satellite imagery data

Raster array format: GeoTiff and HDF format

Land, Ocean, Atmosphere data from spacecraft

MODIS Land Surface Temperature
Geospatial Data Frameworks

- Classic - single machine DBMS or GIS tools
Geospatial Data Frameworks

• Single machine solutions suffer from the **scalability issue**
• In Database community, something is happening..
  • Parallel execution
  • In-memory computation

![Diagram showing one machine with main memory and disk, compared to many machines with multiple cores.]
New DBMS Approaches

• Parallel execution
• GPU acceleration
Cluster (Distributed) Computing Approaches

- Hadoop
- Map Reduce
- Spark
- Apache Flink
- Apache Storm
Manage Spatial Data in Spark?

I want to manage spatial data in Spark!

- No spatial data type support
- No spatial index
- No spatial query

Not that easy!
Outline

- Big geospatial data
- Manage spatial data
- Manage Spatio-Temporal Data
- Spatial Data Analytics in Spark
Manage Spatial Data

- Spatial data partitioning
- Spatial indexing
- Spatial queries
- Optimization
- Language, spatial object support
Spark in a Nutshell

- Resilient Distributed Dataset
- Intermediate data in-memory
- Directed Acyclic Graph (DAG) scheduler
- Spark SQL / DataFrame
- Spark Structured Streaming
- Spark GraphX / GraphFrame
Spark in a Nutshell

- **Action / Transformation**
  - Action: Count, Take
  - Transformation: yield new RDD, such as map, filter, reduce, join, GroupBy

- **Narrow dependency: Map, filter**
  - Parent RDD partition used by 1 child RDD partition
  - No data shuffle, Pipeline execution

- **Wide dependency: Reduce, GroupBy, Sort**
  - Parent RDD partition used by >1 child RDD partition
  - Data shuffle, generate DAG stage
Spatial in Spark: Design Goal

- Reduce wide dependencies
- Speed up local computation
- Reduce the memory footprint
Manage Spatial Data

- Spatial data partitioning
- Spatial indexing
- Spatial queries
- Optimization
- Language, spatial object support
Load Data Into Spark RDD

- Loading data into Spark RDD or DataFrame
- Partition data into 64 MB chunks using Hash partitioner
- If the data is already partitioned, keep the original partitions

https://spark.apache.org/docs/2.3.1/api/java/org/apache/spark/HashPartitioner.html
Spatial Data Partitioning

• Repartition data in RDD
  • Partition by spatial proximity
  • Still achieve load balance
  • API: CustomPartitioner

Spatial Data Partitioning

- Spatial partitioning algorithm
- Randomly sample the RDD
- Build a KD-Tree/Quad-Tree/R-Tree on the sample
- Take the leaf nodes of the tree as the global partition file
- Re-partition the RDD according to the partition file
Spatial Data Partitioning

- Common spatial partitioning grids
- Space partition: Uniform, KD-Tree, Quad-Tree
- Data partition: R-Tree, an overflow partition due to sampling

Spatial Data Partitioning

- Other common spatial partitioning grids
  - Voronoi diagram, Z-curve, Hilbert-curve

Magellan: https://github.com/harsha2010/magellan

Spatial Data Partitioning

- Objects that intersect many boundaries
- Duplicate them to all intersected partitions
- Need duplicate removal after queries
Spatial Data Partitioning

- DAG and data shuffle:
  - Each spatial partitioning is a wide dependency
  - Wide dependency will incur a data shuffle
Spatial Data Partitioning

- Performance
  - Measured using spatial join query
  - Join with 171 thousand polygons
    - NYCtaxi: 1.3 billion points
    - OSMObject: 263 million polygons
    - TIGERedges: 72.7 million line strings
  - Cluster settings: Four workers, one master, 192 cores, 400 GB Memory
Manage Spatial Data

- Spatial data partitioning
- Spatial indexing
- Spatial queries
- Optimization
- Language, spatial object support
Spatial Indexing

- Traditional indexing
  - Not work because of the huge storage overhead
  - Data in different partitions
- Distributed spatial indexing
  - Global index
  - Local index
Spatial Indexing

- Global index
  - Remember the tree built for spatial partitioning?
    - Two birds, one stone!
  - Use it to index partition bounding boxes
  - Lightweight, on the master machine
  - No entries for individual records
Spatial Indexing

• Local indexing
  • On each RDD partition
  • R-Tree, Quad-Tree,…
  • Has entries for individual records
• Queries that use spatial index requires a refinement phase based on the real shapes of objects
Spatial Indexing

- Partition range index (Spatial Hippo, spatial bloom filter)
- Global index only indexes bounding boxes not internal content
- Queries sometimes still go to false positive partitions


Spatial Indexing

- Partition range index (Spatial Hippo, spatial bloom filter)
- On master machine, a concise histogram for each partition
- Reduce false positive partitions
Spatial Indexing

- DAG and data shuffle: 1 RDD transformations
- Global indexing: done with the spatial data partitioning (including partition range index)
- Local indexing: Map per Partition, Narrow dependency
Spatial Indexing

- Performance on different local indexes
- Measured using spatial range query
- Range area from 1% to 16%
- OSMObject: 263 million polygons
- Cluster settings: Four workers, one master, 192 cores, 400 GB Memory
Manage Spatial Data

- Spatial data partitioning
- Spatial indexing
- Spatial queries
- Optimization
- Language, spatial object support
Spatial Queries

- Spatial queries should utilize spatial partitioning and spatial indexing.
- Cache the indexed spatial partitioned RDD.
- The cached RDD cannot be updated. It is expected to be used many times.
Spatial Queries

- Spatial range query: a straightforward way

Send the query window to all partitions

Local index

Local index

Local index
Spatial Range Query

- Prune partitions based on the global index, on master machine
- Prune partitions using partition range index, on master machine
- Go to partitions and check local indexes
- API: rdd.PartitionPruningRDD

Spatial Range Query

- DAG and data shuffle: 1 RDD transformations
  - Checking global indexing -> on master machine
  - Checking local indexing -> a MapPartition operation, no shuffle

![Diagram showing a single stage with RDDs and results, indicating narrow dependency and no shuffle]
Load Spatial Data in Batches

- You are generally tight on memory budget
- Spark needs a great deal of memory
- Use a sliding window to load spatial data in batches
- Sliding window: size = num of partitions, decide it based on mem

Load Spatial Data in Batches

- Use a sliding window to load spatial data in batches
- Load a partition only if its bounding box overlaps query predicate
Spatial Join Query

- A set of objects (gas station), a set of polygons (state boundaries)
- Find gas stations in each state
Spatial Join Query

- Distance join query, similar to spatial join
- Find gas stations within 1 mile distance of each grocery
- Add distance buffer to each grocery = spatial join query
Spatial Join Query

- Algorithm
  - ZipPartition
  - Local index-nested loop join
  - Local de-duplication using the reference point


Dittrich, J-P., and Bernhard Seeger: "Data redundancy and duplicate detection in spatial join processing." In *ICDE*, 2000.
Spatial Join Query Algorithm

- ZipPartition
- Both RDDs should be partitioned in the same way
- One can have local index
Spatial Join Query Algorithm

- Local index-nested loop join

- Local de-duplication using the reference point
  - Spatial partitioning introduces duplicates
  - Need to remove them without incurring data shuffle!

**RDD**

- State
  - California
  - Arizona
  - NYC

**Index**

- Gas

**ForEach** (state) in local states
  Search local index on gas station
Spatial Join Query Algorithm

- Reference point
- Query results with duplicates
  - (Pa, Pb) (Pa, Pb) (Pa, Pb) (Pa, Pb)
- Compute the intersection of Pa and Pb
- Take Reference Point(maxX, maxY) of intersection
- Report (Pa, Pb) in a partition only if reference point is within the boundary of this partition
Spatial Join Query

- DAG and data shuffle: 3 RDD transformations
Spatial K-Nearest Neighbor

• Selection + Sorting phase

Spatial K-Nearest Neighbor

• DAG and data shuffle: 2 RDD transformations
  • Local Top K selection: Map operation, Narrow dependency
  • Global sorting: Wide dependency
Spatial K-Nearest Neighbor

- Why not use global index to prune partitions?
  - Query accuracy is not guaranteed
  - KNN might be in other partitions
- The correct spatial partitioning for KNN should have a K-element buffer and repartition RDD for every KNN query

Too expensive

Some results might be in Arizona partition!
Spatial KNN Join

- Find the nearest 3 gas stations for each grocery
Spatial KNN Join

• Spatial partitioning: a distance buffer for each partition such that each query point can find its KNN in one RDD partition.

• Local KNN


Spatial KNN Join

• DAG and data shuffle: 2 RDD transformations

Diagram showing the process of Spatial KNN Join with RDDs for California, Arizona, and NYC, with separate Index RDDs for each state. The diagram also highlights narrow dependencies and network communication.
Manage Spatial Data

- Spatial data partitioning
- Spatial indexing
- Spatial queries
- Optimization
- Language, spatial object support
Optimization

- Query optimization
- Distributed spatial join VS broadcast spatial join
  - One side of spatial join is smaller, send it to all RDD partitions
Optimization
Custom Serializer

• What is a serializer?
  • Object -> byte array -> Object

• When is a serializer used?
  • Cache RDD into memory
  • Shuffle objects across the cluster
Optimization
Custom Serializer

• Why do we need a custom serializer for spatial objects?
  • Spatial objects are very complex, tons of coordinates
  • Spark default Java and Kryo serializer are not efficient
  • Size according to GeoSpark experiment
    • 3 times smaller than Spark default size
    • 20 times faster serialization
    • 5 times faster deserialization

Optimization
Custom Serializer

• How to write a spatial object serializer?
  • Define a rule to serialize heterogeneous spatial types into a byte array. For example, borrow the definition of Shapefile or WKB

• How to write a spatial index serializer?
  • Use a regular tree traversal algorithm to traverse the tree
  • Note the child node size because an index is not a full tree
Optimization

Custom Serializer

• How to add a serializer to Spark?
• Write a register via Kryo
• Register it when creating Spark session

```java
var sparkSession = SparkSession.builder()
  .appName("myAppName")
  // Enable GeoSpark custom Kryo serializer
  .config("spark.serializer", classOf[KryoSerializer].getName)
  .config("spark.kryo.registrator", classOf[GeoSparkKryoRegistrator].getName)
  .getOrCreate()
```

Manage Spatial Data

- Spatial data partitioning
- Spatial indexing
- Spatial queries
- Optimization
- Language, spatial object support
**Language**

- Implement the system in what language?
  - Scala
    - Spark is written in Scala
    - Functional programming by nature
  - Java
    - No learning curve, Scala/Java functions can call each other
    - Cannot modify Spark kernel
    - Cannot add UserDefinedType and query optimization
  - Python
    - Python code connect to Spark via Py4j
    - Needs Python spatial object handler
Spark Interface

- Spark interface
  - RDD: easy to customize, hard to use
  - DataFrame: easy to use, hard to customize
    - Spatial SQL
    - User Defined Type
    - Indexing and spatial partitioning
    - Optimized join strategy
Spark Catalyst query optimizer

Integrate With Dataframe

SQL

DataFrames

Spatial SQL Expressions

System Catalog

Spatial SQL Heuristic Rules

Unresolved Logical Plan

Analysis

Logical Plan

Logical Optimization

Optimized Logical Plan

Physical Plans

Cost Models

Selected Physical Plan

Code Generation

Dataframe

Physical Planning

Cost-based Spatial Join Strategies

Spatial Statistics

Make Catalyst understand the geospatial!
Integrate With Dataframe
Spatial SQL

• Spatial SQL: SQL-MM3, OGC Simple Feature Access
  • SQL-MM3: PostGIS, GeoSpark, GeoMesa…
    • ST_Contains, ST_Within
  • OGC Simple Feature Access
    • Contains, Within
• Compatible with each other in most cases
• Implement these functions in Spark expression (not UDF)
Integrate With Dataframe

Spatial SQL

- Spark expression, the way Spark writes its own functions
- Unary, binary, ternary
- Each AST (Abstract Syntax Tree) node is a Spark expression
- Allow the following features
  - Code generation
  - Output data type
  - Fuse into the Catalyst optimizer

Integrate With Dataframe

User Defined Type

• Each spatial object and index must be a UDT in Dataframe
• Spark provides a developer API
• The spatial object UDT must be based on a primitive type: Array
• Must provide the serialization method

```
private[sql] class GeometryUDT extends UserDefinedType[Geometry] {
  override def sqlType: DataType = ArrayType(_byteType, containsNull = false)

  override def userClass: Class[Geometry] = classOf[Geometry]

  override def serialize(obj: Geometry): GenericArrayData = {
    new GenericArrayData(GeometrySerializer.serialize(obj))
  }

  override def deserialize(datums: Any): Geometry = {
    datums match {
      case values: ArrayData => {
        return GeometrySerializer.deserialize(values)
      }
      case _ => {
        throw new Exception("Not an array")
      }
    }
  }

  case object GeometryUDT extends GeometryUDT
}
```

UDT code snippet from GeoSpark
Integrate With Dataframe
Indexing and Spatial Partitioning

- **Indexing**
  - Each local index is a big UDT. Each partition has one UDT (row).
  - No way to plug the global index to Catalyst physical plan

- **Spatial partitioning**
  - Cannot be done via regular DataFrame API
  - Use DataFrame’s RDD API to do spatial partitioning
Integrate With Dataframe

Optimized Spatial Join Strategy

• Inject spatial join query
• Overwrite Spark strategy (physical plan)
• Use pattern-matching to capture spatial join pattern

Join pattern-matching snippet from GeoSpark

```python
from GeoSpark

def apply(plan: LogicalPlan): Seq[SparkPlan] = plan match {
  case Join(left, right, Inner, SomeST_Contains(Seq(leftShape, rightShape)))) =>
    planSpatialJoin(left, right, Seq(leftShape, rightShape), false)
  case Join(left, right, Inner, SomeST_Intersects(Seq(leftShape, rightShape)))) =>
    planSpatialJoin(left, right, Seq(leftShape, rightShape), false)
  case Join(left, right, Inner, SomeST_Within(Seq(leftShape, rightShape)))) =>
    planSpatialJoin(left, right, Seq(rightShape, leftShape), true)
  case Join(left, right, Inner, SomeST_Overlaps(Seq(leftShape, rightShape)))) =>
    planSpatialJoin(left, right, Seq(rightShape, leftShape), false)
  case Join(left, right, Inner, SomeST_Touches(Seq(leftShape, rightShape)))) =>
    planSpatialJoin(left, right, Seq(leftShape, rightShape), true)
  case Join(left, right, Inner, SomeST_Distance(Seq(leftShape, rightShape), radius)) =>
    planDistanceJoin(left, right, Seq(leftShape, rightShape), radius, true)
  // More cases...
}
```
Integrate With Dataframe
Optimized Spatial Join Strategy

• Register the new join strategy using its

• `sparkSession.experimental.extraStrategies = JoinQueryDetector`

Captured join query plan

```
SELECT *
FROM polygondf, pointdf
WHERE ST_Contains(polygondf.polygonshape, pointdf.pointshape)
```

Original join query plan

```
SELECT *
FROM polygondf, pointdf
WHERE ST_Contains(polygondf.polygonshape, pointdf.pointshape)
```

== Physical Plan ==

```
RangeJoin

<table>
<thead>
<tr>
<th>polygonshape#20: geometry, pointshape#43: geometry, false</th>
</tr>
</thead>
<tbody>
<tr>
<td>:- Project [st_polygonfromenvelope(XXX) AS polygonshape#20]</td>
</tr>
<tr>
<td>: +- *FileScan csv</td>
</tr>
<tr>
<td>+- Project [st_point(XXX) AS pointshape#43]</td>
</tr>
<tr>
<td>+- *FileScan csv</td>
</tr>
</tbody>
</table>
```

```
BroadcastJoin

<table>
<thead>
<tr>
<th>polygonshape#20: geometry, pointshape#43: geometry, false</th>
</tr>
</thead>
<tbody>
<tr>
<td>:- Project [st_polygonfromenvelope(XXX), mypolygonid) AS polygonshape#20]</td>
</tr>
<tr>
<td>: +- *FileScan csv</td>
</tr>
<tr>
<td>+- Project [st_point(XXX) AS pointshape#43]</td>
</tr>
<tr>
<td>+- *FileScan csv</td>
</tr>
</tbody>
</table>
```
Outline

Big geospatial data

Manage spatial data

Manage Spatio-Temporal Data

Spatial Data Analytics in Spark
Spatial Visual Analytics

- Spatial visualization is important
- Existing tools can exhibit excellent visual effects but cannot scale
Spatial Visual Analytics

- Scalable visualization: visualize BILLION objects on Gigapixel map
- Customizable visualization: manipulate pixels at scale
Spatial Visual Analytics

• Rasterize vector shapes to pixels (with weights)
  • ST_Pixelize
  • Or, load from GeoTIFF/NetCDF/HDF: array-format spatial observations
• Aggregate pixels (with weights)
• Render color - ST_Render

Geotrellis: https://geotrellis.io/
Pixel Array Data Partitioning

• Each partition is a map tile
• Each map tile is X*X pixel array
RDD and Zoom Levels

- Zoom levels: each level consists of a set of map tiles
- Each RDD is a zoom level.

<table>
<thead>
<tr>
<th>Level</th>
<th>#Tiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Map Visualization Pipeline

- Visualization pipeline and DAG stages

NYC Taxi Trips
- Pickup Location
- Drop-off Location
- Fare
- Tips
- Landmarks (airports, schools, ...)
- Boundary
  - Landmark name
  - Address

Spatial join query
- Repartition
- Local join
- Remove Duplicates

Rasterize

Pixel aggregate

Colorize

Render

Stage 1
Stage 2
Stage 3
Stage 4
Manipulate Raster Array Data

• Map algebra operations

Local operations

Focal operations

Zonal operations

Local and focal gif from Geotrellis: https://docs.geotrellis.io/en/latest/guide/core-concepts.html#map-algebra
A Map Algebra Example

- Local operation on temperature observations from NASA MODIS

Figures and examples from https://gisgeography.com/map-algebra-global-zonal-focal-local/
Local Operation

- **Algorithm:** two pixel RDDs A and B, partitioned in the same way
- **ZipPartitions:** Zip A and B
- **MapPartition:** Local pixel manipulation

<table>
<thead>
<tr>
<th>Stage</th>
<th>RDD</th>
<th>Tile1</th>
<th>Tile2</th>
<th>Tile3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RDD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MergedT1</td>
<td>MergedT2</td>
<td>MergedT3</td>
<td></td>
</tr>
</tbody>
</table>

Narrow dependency
Focal Operation

- Algorithm: each pixel aggregates with its neighbors
- Spatial partition buffered Pixels
- Make sure each pixel can find its neighbors
- MapPartition: Local aggregation in each partition

![Diagram showing pixel aggregation and spatial partition](image-url)
Zonal Operation

- **Algorithm 1**: Join vector polygons with raster pixels
  - Rasterize each polygon to a mask layer
  - Broadcast it to each pixelRDD partition
  - Find matched pixels on each partition
  - Similar to a range query. Loop every polygon, not scalable
Zonal Operation

- Algorithm 2: Scalable
  - Convert each pixel back to a spatial point
  - Then use spatial join between polygons and points

```
<table>
<thead>
<tr>
<th>Stage</th>
<th>RDD</th>
<th>RDD</th>
<th>RDD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>California</td>
<td>Arizona</td>
<td>NYC</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td></td>
<td>State</td>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>Gas</td>
<td>Gas</td>
<td>Gas</td>
</tr>
</tbody>
</table>
```

Narrow dependency
Spatial Data Mining Example

• Spatial co-location pattern mining in Spark
  • Use spatial join to build a whole data mining application
  • Use map algebra to visualize the result
• Data: Taxi pick ups (1 billion)
• Data: NYC landmarks (300, airports, hospitals..)

https://github.com/jiayuasu/GeoSparkTemplateProject/tree/master/geospark-analysis
What Is Spatial Co-Location

- Two or more species are often located in a neighborhood relationship. Africa lions co-locates with zebras.
- Ripley’s K function is often used in judging co-location.
  - Executes multiple times
    - Compute adjacent matrix (distance join)
    - Get a K score
  - Form a curve for observation
Ripley's K Function
Multivariate Spatial Patterns

1. Set a base distance (say, 1 meter)
2. Perform a distance join to get adjacent matrix between two species
3. Plug the matrix into Ripley’s K and compute the K value
4. Repeat Step 2 and 3 until converge
   - Each time increase the distance

Write Co-Location Mining in Spark

- Create TripRDD (PointRDD)
  - Spatial partition
  - Build index
- Cache indexed TripRDD into Spark memory
- Create LandmarkRDD (PointRDD)
  - Do not do spatial partitioning for now

```scala
tripRDD.spatialPartitioning(GridType.KDBTREE)
tripRDD.buildIndex(IndexType.QUADTREE, true)
tripRDD.indexedRDD = tripRDD.indexedRDD.cache()
```
Write Co-Location Mining in Spark

- Start iterations
  - Create a CircleRDD = LandmarkRDD + distance buffer
  - Spatial partition CircleRDD in the way with TripRDD
  - Perform distance join
  - Compute Ripley’s K

```scala
var bufferedArealmRDD = new CircleRDD(arealmRDD, currentDistance
bufferedArealmRDD.spatialPartitioning(tripRDD.getPartitioner)

var adjacentMatrix = JoinQuery.DistanceJoinQueryFlat(tripRDD,
bufferedArealmRDD, true, true)
```
Write Co-Location Mining in Spark

- Mining result
  - Observed K value is always higher than expected K value
  - Conclusion: people call taxis at landmarks such as airport, hospital, library…

![Ripley's L function using 10 iterations](image.png)
Write Co-Location Mining in Spark

- DAG and data shuffle

Stage 1
- Trip
  - Partition
- Landmark
  - Partition

Stage 2
- Spatial partition
- Indexing

Stage 3
- ZipPartition
- Local join
- De-dup
- Ripley’s K

Iterations for Ripley’s K
Write Co-Location Mining in Spark

- Visual analytics
- TripRDD
- LandmarkRDD
- Map algebra: local operation
Outline

- Big geospatial data
- Manage spatial data
- Manage Spatio-Temporal Data
- Spatial Data Analytics in Spark
Manage Spatial-Temporal Data

• What is spatial-temporal data?

• What is a spatial-temporal query?

```
SELECT *
FROM tweets t
WHERE ST_Contains(t.loc, US) AND timestamp BETWEEN 11/1/2017 AND 11/30/2017
```

• Why do we need to care spatial-temporal data?
  • Temporal filter is done in a table scan. Inefficient!
  • Spatial data distribution / shape changes over time (Trajectories!)

### Example

<table>
<thead>
<tr>
<th>Twitter</th>
<th>Location</th>
<th>Timestamp</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>3</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Spatial-Temporal Partitioning

• Partition by spatial and temporal proximity / achieve load balance
• Randomly sample the RDD and put it on the master
• Build the global index / partition boundaries on the sample
• Apply partitions….

• How to partition data by spatial and temporal attributes together?
Spatial-Temporal Partitioning

- Temporal partitioning
- Uniform granularity
- Load-balanced

Compute Temporal and Spatial partitions separately

Spatial-Temporal Partitioning

- Spatial partitioning
  - Compute the spatial boundaries using KD-Tree, Quad-Tree,…

Compute Temporal and Spatial partitions separately
Spatial-Temporal Partitioning

- What if the spatial data distribution changes over time?

Spatial-Temporal Partitioning

- First, generate temporal partitions on the sample
- Then, create spatial partitions for each temporal partition
- Local spatial index is still built on each spatial-temporal partition
Spatial-Temporal Queries

- Spatial-temporal range query
  - Global index: temporal filter, then spatial filter
  - Prune partitions
  - Query remaining local indexes
- Spatial-temporal join query
  - Partition both datasets in the same way
  - Zip partitions by ID
  - Local join
Wrap-Up
Manage spatial data

- Spatial partitioning
- Spatial indexing
- Spatial queries

Manage Spatio-Temporal Data

- Spatial-temporal partitioning

Spatial Data Analytics in Spark

- Distributed map visualization
- Distributed map algebra
- Spatial co-location pattern mining

Query optimization
- Object serializer
- Spark integration