Big Spatial Data Management on Spark
Tons of Spatial data out there...

Geotagged Microblogs

Geotagged Pictures

Medical Data

Smart Phones

Sensor Networks

VGI

Satellite Images

Traffic Data
Beast

• A Spark add-on for Big Exploratory Analytics on Spatio-Temporal data
• Developed at UCR
  ▪ You will get high-quality support 😊
• Already used in UCR-Star and other live applications
Geometry Data Types

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection
Geometry Predicates

A Contains B
A Overlaps C
B Disjoint C
A Touches D
Geometric Analysis Functions

• Create Point, LineString, ...
• Intersection, Union, Difference
• Area, Length
• Centroid, Convex Hull
Spatial Feature (IFeature)

Feature = Geometry + Other Attributes

- Example
  - Road(Geometry, Name, Speed Limit)
  - State(Geometry, Name, Population)

- SpatialRDD = RDD[IFeature] or JavaRDD<IFeature>
Data Source

- **UCRStar.com**
- 200+ datasets
- Full/subset download
- Standard formats

- **Spider.cs.ucr.edu**
- Still beta
- Data generator
Spatial Functions in Spark

• Data loading
• Simple manipulation
• Summarization
• Partitioning
• Range filters
• Spatial join
• Visualization
Project Setup

**pom.xml**

```xml
<dependencies>
  <dependency>
    <groupId>edu.ucr.cs.bdlab</groupId>
    <artifactId>beast-spark</artifactId>
    <version>0.8.2</version>
  </dependency>
</dependencies>
```

**App.scala**

```scala
import edu.ucr.cs.bdlab.beast._
```
// Load a shapefile
val polygons: RDD[IFeature] = sc.shapefile("tl_2018_us_state.zip")

// Load GeoJSON file
val points = sc.geojsonFile("Tweets.geojson")

// Load points from a CSV file
val lines = sc.readCSVPoint("Crimes.csv", "Longitude", "Latitude", ",", skipHeader = true)

// Load geometries from a CSV file
val lines = sc.readWKTFile("States.csv", 0, \
    \t, skipHeader = false)
// Calculate the area and append as a new attribute
polygons.map(f => {
    val area = f.getGeometry.getArea
    val newF = new Feature(f)
    newF.appendAttribute("area", area)
    newF
})

// Simplify the geometries into their convex hull
polygons.map(f => {
    val ch = f.getGeometry.convexHull()
    val newF = new Feature(f)
    newF.setGeometry(ch)
    newF
})
// Calculate a simple summary for geometries
val summary: Summary = polygons.summary
println(summary)

Output

MBR: [(-179.231086, -14.601813), (179.859681, 71.439786)], size: 14807211, numFeatures: 56, numPoints: 924434, avgSideLength: [12.18881225, 4.2761075]
// Calculate a histogram of 100 x 100
val histogram = points.uniformHistogramCount(Array(100, 100))
println(histogram.getValue(Array(0, 0), Array(40, 10)))

Output
482
Spatial Partitioning

// Partition the dataset into 100 partitions using a uniform grid partitioner
val partitionedPoints: RDD[(Int, IFeature)] =
points.partitionBy(classOf[GridPartitioner], 100)

// More balanced partitions
val partitionedPoints: RDD[(Int, IFeature)] =
points.partitionBy(classOf[RSGrovePartitioner], 100)
// Select the geometry of the state of California
val california: IFeature = polygons.filter(f =>
f.getAttributeValue("NAME") == "California").first()

// Filter the points that are inside the state of California
val californiaPoints =
points.rangeQuery(california.getGeometry)
println(s"Number of points in California
${californiaPoints.count()}")

Output

Number of points in California 259657
// Count points per state
val airportCountByState = polygons.spatialJoin(airports)
  .map(fv => (fv._1.getAttributeValue("NAME"), 1))
  .countByKey()

airportCountByState.foreach(sv =>
println(s"${sv._1}
\t${sv._2}"))

Output
New Mexico 1
Connecticut1
Commonwealth of the Northern Mariana Islands 2
California 12
Nevada 3
Visualization

// Plot states as an image
polygons.plotImage(2000, 2000, "states.png")
Visualization on a Map

// Plot states as a multilevel map
polygons.plotPyramid("states", 10,
opts = "mercator" -> "true")
Writing the output

// Save the output as a decompressed shapefile
polygons.saveAsShapefile("output.shp")
// Save the output as a GeoJSON file
polygons.saveAsGeoJSON("output.geojson")
// Save as a WKT file
polygons.saveAsWKTTFile("output.tsv", 0, '\t'
// Save points as a CSV file
polygons.saveAsCSVPoints("output.csv", 0, 1, ',
// Save as KML file
polygons.saveAsKML("output.kml")
Other Big Spatial Data Systems

• Apache Sedona (Formerly GeoSpark)
  ▪ Developed at ASU
  ▪ In incubation
    [http://sedona.apache.org]
• PySAL [https://pysal.org]
  ▪ For Python users
  ▪ Maintained by the Center for Geospatial Sciences at UCR
Summary

• There are tons of big spatial data
• Beast can help you processing big spatial data in Spark such as:
  ▪ Loads data in standard formats
  ▪ Manipulates feature attributes
  ▪ Summarizes the data
  ▪ Filters by range
  ▪ Joins multiple datasets
  ▪ Visualizes the results
Further Readings

• Beast Wiki Pages
  ▪ https://bitbucket.org/eldawy/beast/wiki/Home

• Code Examples
  ▪ https://bitbucket.org/eldawy/beast-examples/src/master/

• Visualization Paper