Once upon a time...
Claudius Ptolemy (AD 90 – AD 168)
Al Idrisi (1099–1165)
Cholera cases in the London epidemic of 1854
Figure 3—Children under 15 years of age in 1940.
Figure 3—Children under 15 years of age in 1940.
Cool computer technology..!! Can I use it in my application?

I have BIG data. I need HELP..!!

Oh..!! But, it is not made for me. Can’t make use of it as is

My pleasure. Here it is.

I have BIG data. I need HELP..!!
Kindly let me understand your needs

1969

Kindly let me get the technology you have
HELP...!! I have BIG data. Your technology is not helping me.

HELP...!! I have BIG data. Your technology is not helping me.

mmmm... Let me check with my good friends there.

Cool Database technology...!! Can I use it in my application?

Oh...!! But, it is not made for me. Can’t make use of it as is.

My pleasure. Here it is.
Kindly let me understand your needs.

Kindly let me get the technology you have.
HELP..!! Again, I have **BIG** data. Your technology is not helping me.

Cool **Big Data** technology..!! Can I use it in my application?

Let me check with my other good friends there.

Sorry, seems like the DBMS technology cannot scale more.

Oh..!! But, it’s not made for me. Can’t make use of it as is.

Oh..!! But, it’s not made for me. Can’t make use of it as is.

My pleasure. Here it is.

Can I use it in my application?

My pleasure.

Here it is.
Kindly let me understand your needs.
Big Spatial Data Management
Tons of Spatial data out there…

Geotagged Microblogs

Geotagged Pictures

Medical Data

Smart Phones

Sensor Networks

VGI

Satellite Images

Traffic Data
Spatial Data on Spark

```scala
val points: RDD[(Double, Double)] = sc.textFile("points.csv")
  .map(l => {
    val coords = l.split(",").map(_.toDouble)
    (coords(0), coords(1))
  })
val xmin, ymin, xmax, ymax: Double = null
val result = points.filter(point => {
  point._1 >= xmin && point._1 < xmax &&
  point._2 >= ymin && point._2 < ymax
})
result.map(pt => s"${pt._1},${pt._2}"
  .saveAsTextFile("output")

val points: RDD[IFeature] = sc.readCSVPoint("points.csv")
val range = new GeometryFactory().toGeometry(
  new Envelope(xmin, xmax, ymin, ymax))
val results = points.rangeQuery(range)
results.saveAsCSVPoints("output.csv")
```

193 seconds

2 seconds
The Built-in Approach of Beast

**The On-top Approach**
- Spatial Modules
- User Programs
- SQL
- Spark Java/Scala APIs
- Job Monitoring and Scheduling
- RDD Runtime
- Storage (HDFS)

**From Scratch Approach**
- (Spatial) User Program
  - + RDD APIs
  - + Job Monitoring and Scheduling
  - + RDD Runtime
  - + Storage
  - + ...

**The Built-in Approach (Beast)**
- User Programs
- Spatial Language
- SQL
- Spark Java/Scala APIs
- Spatial Operators
- Early Pruning
- Spatial Indexing
- Job Monitoring and Scheduling
- RDD Runtime
- Storage (HDFS)
Domain-specific Big-data

• Spark and similar frameworks are general purpose systems
• They can be customized for a specific domain
• This part is an example of how to customize a big-data system for the domain of spatial data
Beast Architecture

Big Spatial Data Apps

Visualization Framework
RDD-based Query Processor
Spatial Partitioner & Load Balancer
In-situ Spark Loaders/Writers
Spatial Data Types
Beast Architecture

Visualization Framework

RDD-based Query Processor

Spatial Partitioner & Load Balancer

In-situ Spark Loaders/Writers

Spatial Data Types
Spatial Data Types

- RDD is flexible enough to allow any user-defined class to be used with RDD
- In Beast, we define the following types
  - Point: n-dimensional point
  - Envelope: n-dimensional box
  - Geometry: Any vector-based geometry
  - Feature: Geometry + attributes
Spatial Data Types

- Point
- Envelope
- Geometry
- Feature
import org.apache.spark.rdd.RDD
import edu.ucr.cs.bdlab.beast.geolite.IFeature
val buildings: RDD[IFeature] = sc.geojsonFile("buildings.geojson")

val polygons: SpatialRDD = sc.shapefile("us_counties")
val randomPoints: SpatialRDD = sc.generateSpatialData.
mbr(polygons.summary).uniform(1000000)
val sjResult = polygons.spatialJoin(randomPoints)
val counties: SpatialRDD = sc.shapefile("us_counties")
counties.toDataFrame(spark).createOrReplaceTempView("counties")
val counties_areas = spark.sql(
  "SELECT NAME, g, ST_Area(g) FROM counties"
)counties_areas.toSpatialRDD.saveAsGeoJSON("us_counties_areas")

import edu.ucr.cs.bdlab.beast.indexing.RSGrovePartitioner
val partitioned: RDD[(Int, IFeature)] = sc.shapefile("points.shp").
  partitionBy(classOf[RSGrovePartitioner])
partitioned.saveAsIndex("partitioned_data", "shapefile")

// To load the data back in another Spark application
val loadedPartitioned = sc.shapefile("partitioned_data")

sc.shapefile("us_counties")
   .plotImage(2000, 2000, "counties.png")
Beast Architecture

- Visualization Framework
- RDD-based Query Processor
- Spatial Partitioner & Load Balancer
- In-situ Spark Loaders/Writers
- Spatial Data Types
Spark Loaders

• In Spark, a data loader is a top-level RDD that does not depend on any other RDD

• To load data in an input path:
  ▪ Define partitions based on the input metadata
  ▪ Provide a parser for one partition that extracts all records
Spatial binary files

Input file

Header

Data records

RDD[IFeature]

Header

Partition 1

Partition 2

Partition n

Boundary records
A record is processed by the partition that contains its start offset

Load

compute

Iterator[IFeature]
Spark Writer

• Implemented as an action
• Operates on RDD[IFeature] and writes all its contents to an output path
• Each partition is written to a separate file
Spark Writer

RDD[IFeature]

Partition 1

Partition 2

…

Partition n

HDFS

Part 1

Part 2

…

Part n
Beast Architecture

Big Spatial Data Apps

- Visualization Framework
- RDD-based Query Processor
- Spatial Partitioner & Load Balancer
- In-situ Spark Loaders/Writers
- Spatial Data Types
Data Loading in HDFS

- Blindly chops down a big file into 128MB chunks
- Values of records are not considered
- Relevant records are typically assigned to two different blocks
- HDFS is too restrictive where files cannot be modified
Two-layer Index Layout
Uniform Grid

Works only for uniformly distributed data
R-tree

• Read a sample
• Partition the sample using an R-tree index
• Use MBR of leaf nodes as partition boundaries for all the data
R-tree

- Read a sample
- Partition the sample using an R-tree index
- Use MBR of leaf nodes as partition boundaries for all the data
R-tree-based Index of a 400 GB road network
Non-partitioned dataset
Spatially Partitioned RDD

- RDD + Partitioner
- Spark allows custom partitioners
Index Writing and Loading

• Beast provides an option to write an index to disk and read it back
• This gives an option to load an already partitioned RDD
Index Writing and Loading

RDD[IFeature]

Spatial Partitioner

Partition 1

Partition 2

...

Partition n

 Partition 1

Partition 2

...

Partition n

Spatial Partitioner

.master-file

Part 1

Part 2

...

Part n

HDFS

Write

Load

RDD[IFeature]

Partition 1

Partition 2

...

Partition n
Beast Architecture

Big Spatial Data Apps

Visualization Framework

RDD-based Query Processor

Spatial Partitioner & LoadBalancer

In-situ Spark Loaders/Writers

Spatial Data Types
RDD Processing

• Since a spatial RDD is just a regular RDD, all existing transformations and actions can work seamlessly on it

• In addition, we have specialized handling for spatial queries
  ▪ Range Query
  ▪ Spatial Join
Range Query

Use the partition information to prune disjoint partitions

Scan matching partitions in parallel to find matching records
Range Query

RDD[IFeature]

Spatial Partitioner

Partition 1

Partition 2

Partition n

PartitionPruning RDD[IFeature]

Spatial Partitioner

Partition 1

Partition n

Filter

RDD[Ifeature]

Spatial Partitioner

Partition 1

Partition n

Question: Is this narrow or wide dependency?
Spatial Join

Join Directly

Partition – Join
Spatial Join

Join Directly

Partition – Join

Total of 36 overlapping pairs

Only 16 overlapping pairs
Join Directly

Question: Is this narrow or wide dependency?
Join Directly

Question: Is this narrow or wide dependency?
Beast Architecture

Visualization Framework

- RDD-based Query Processor
- Spatial Partitioner & Load Balancer
- In-situ Spark Loaders/Writers
- Spatial Data Types
The goal of HadoopViz is not to propose new visualization techniques, instead its goal is to scale out existing techniques.
Heat Map From 2009 to 2014
Month-by-Month
Jan-2009

72 Frames × 14 Billion points per frame
Total = 1 Trillion points
Created in 3 hours on 10 nodes instead of 60 hours
Abstract Visualization

Input Partition

1. smooth

2. Create canvas

3. plot

4. merge

5. write

Output Image
Example: Satellite Data Visualization

1. **Smooth**: Recover holes

2. **Create Canvas**: Initialize a 2D Matrix with zeros

3. **Plot**: Update the matrix

4. **Merge**: Matrix addition

5. **Write**: Generate the image
Example: Road Network Visualization

1. **Smooth**: Merge intersections
2. **Create Canvas**: Create a blank image
3. **Plot**: Draw roads as polygons
4. **Merge**: Plot an image on the other
5. **Write**: Encode as PNG and write to file
Single Level Image

Input

Split

Split

Split

Split

Merge
(Overlay)
Space Partitioning

Input

Split

Split

Split

Split

create-raster

create-raster

create-raster

create-raster

smooth

smooth

smooth

smooth

rasterize

rasterize

rasterize

rasterize

rasterize

merge (Stitch)
Level of Details
Map of California – 2GB
Generated in 2 minutes on 10-node cluster instead of one hour
Multi-level Image

- Many images at different zoom levels
  - Pan
  - Zoom in/out
  - Fly to
- More details as the zoom level increases
- Number of tiles increases exponentially
Multi-level Visualization

- Abstract multi-level visualization algorithm
- The choice of partitioning technique changes for each zoom level

![Diagram of multi-level visualization]

- Default partitioning
- Threshold level $z_\theta$
- Spatial Partitioning
Beast Architecture

Big Spatial Data Apps

Visualization Framework

RDD-based Query Processor

Spatial Partitioner & Load Balancer

In-situ Spark Loaders/Writers

Spatial Data Types
Thank You

Questions?
A Unified Big Data Interface

Unified Big Data Abstraction

- Cost Model
- Query Optimizer
- Query Executor

YARN – Resource Manager

HDFS – File System
Language

Applications: SHAHED [ICDE’15] – MNTG [SSTD’13, ICDE’14◊]
TAREEG [SIGMOD’14◊, SIGSPATIAL’14]

Language
Pigeon [ICDE’14◊]

Visualization
[VLDB’15◊, ICDE’16]

Operations
Basic operations – CG_Hadoop
[SIGSPATIAL’13, TSAS*]

MapReduce
Spatial File Splitter
Spatial Record Reader

Indexing
Grid – R-tree – R+-tree – Quad tree
[VLDB’15]

* Under review
◊Demo paper
Language (Pigeon)

• Hides the complexity of the system with a high level language
• OGC standard used by Oracle Spatial and PostGIS
• Extends Pig Latin with OGC-compliant primitives
  - Spatial data types (e.g., Polygon)
  - Basic operations (e.g., Area)
  - Spatial predicates (e.g., Touches)
  - Spatial analysis (e.g., Union)
  - Spatial aggregate functions (e.g., Convex Hull)

A. Eldawy and M. F. Mokbel, “Pigeon: A Spatial MapReduce Language”, Demo at IEEE ICDE’14
Spatial Data Types

Data Loading

```
lakes = LOAD 'lakes' AS (id:int, area: polygon);
```

Range Query

```
houses_in_range = Filter houses BY Overlap(house_loc, range);
```

KNN

```
nearest_houses = KNN houses WITH_K=100 USING DistanceTo(house_loc, query_loc);
```

Spatial Join

```
lakes_states = Join lakes BY lakes_boundary states BY states_boundary Predicate = Overlap
```
Spatio-temporal Indexing

Applications: SHAHED [ICDE’15] – MNTG [SSTD’13, ICDE’14]
TAREEG [SIGMOD’14, SIGSPATIAL’14]

Language
Pigeon [ICDE’14]

Visualization
HadoopViz [VLDB’15]

Operations
Basic operations – CG_Hadoop
[SIGSPATIAL’13, TSAS*]

MapReduce
Spatial File Splitter
Spatial Record Reader

Indexing
Grid – R-tree – R+-tree – Quad tree
[VLDB’15]

Multiresolution Spatio-temporal Index

Yearly Indexes

2012

jan
feb
... dec
jan
feb
... dec
jan

Monthly Indexes

1 2
... 366 1 2
... 365 1 2
... 31

Daily Indexes

Performance of SHAHED

(c) Selection Query

(d) Visualization
Reference Point

Intersection rectangle

Reference point

r
S
Index building
Index Building for NASA Data

The graph shows the time (in minutes) taken to build an index for different input sizes (TB) of NASA data. The input sizes are 1.2TB, 2TB, and 4.6TB.

- For 1.2TB input, the time is approximately 100 minutes.
- For 2TB input, the time increases to approximately 300 minutes.
- For 4.6TB input, the time rises significantly to over 800 minutes.

The graph indicates that the time required to build an index increases significantly with larger input sizes.
Related Work

• Most techniques for spatial data processing in Hadoop use Hadoop as a blackbox
  ▪ RQ, KNN and SJMR [Zhang et al’09]
  ▪ R-tree construction [Cary et al’09]
  ▪ KNN Join [Lu et al’12, Zhang et al’12]
  ▪ RNN [Akdogan et al’10]
  ▪ ANN [Wang et al’10]

• MD-HBase [Nishimura et al’11]
  ▪ Framework for multi-dimensional data processing
  ▪ Based on HBase, a key-value store on HDFS
  ▪ Does not support MapReduce programming
Map plan – SpatialHadoop

Indexed Input File(s)

Number of splits

Spatial File Splitter

Map task

Split

Spatial Record Reader

k,v

Map

Map task

Split

Spatial Record Reader

k,v

k,v

Map
KNN

SpatialFileSplitter selects the block that contains the query point.

Map function performs kNN in the selected block.

Answer is tested for correctness.

✔ Answer is correct.
KNN

First iteration runs as before and result is tested for correctness

✗ Answer is incorrect

Second iteration processes other blocks that might contain an answer
Range query

![Graph showing performance of different systems with varying file sizes. The y-axis represents throughput (Jobs/Minute), and the x-axis represents file size (GB). The graph compares SHadoop, Hadoop-SJMR, SHadoop-R-tree, SHadoop-R+-tree, SHadoop-Grid, and Hadoop. There are bars indicating 1.4, 1.8, and 1.16 GB sizes.]
K Nearest Neighbor
Preliminary Results