Spark + Simba: Efficient In-Memory Spatial Analytics.

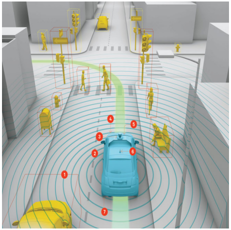
Based on D. Xie, F. Li, B. Yao, G. Li, L. Zhou and M. Guo
SIGMOD'16.

Andres Calderon

April 24, 2018

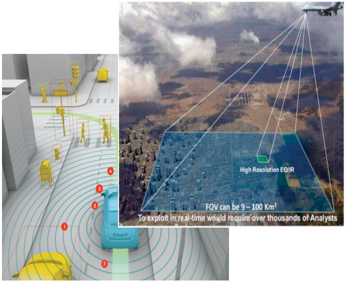
Introduction

- There has been an explosion in the amount of spatial data in recent years...



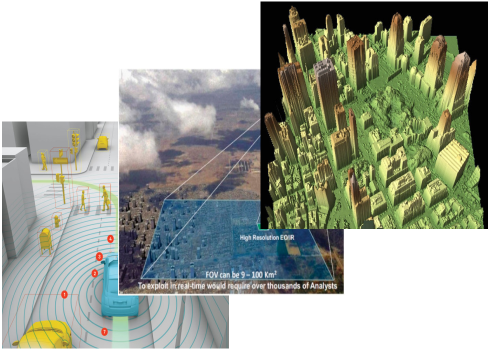
Introduction

- There has been an explosion in the amount of spatial data in recent years...



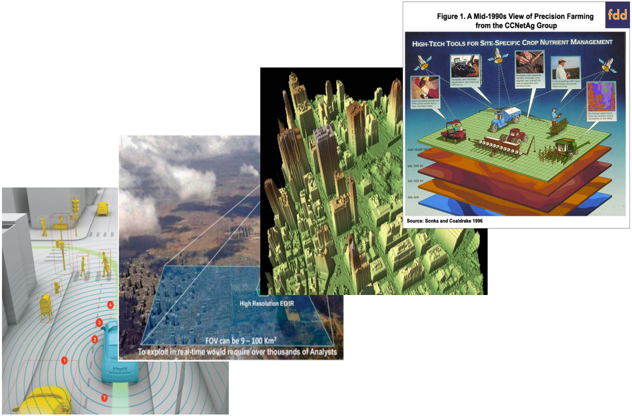
Introduction

- There has been an explosion in the amount of spatial data in recent years...



Introduction

- There has been an explosion in the amount of spatial data in recent years...



Applications

- The applications and commercial interest is clear...

Spatial is Special

- But remember that “Spatial is Special”...

Spatial is Special

- But remember that “Spatial is Special”...

Spatial is Special

- But remember that "Spatial is Special"...

PostGIS
Spatial PostgreSQL
ORACLE
SPATIAL
ArcGIS
ESRI
MySQL
Microsoft SQL Server

Hadoop-GIS
Spatial Big Data Solutions
MD-Hbase
Spatial Hadoop
I can play with your spatial data
SECONDO
geomesa

GeoSpark
GeoTrellis
Spark SQL
SpatialSpark
MEMSQL

Is there room for improvements?

- Why do we need a new tool???

Yes, there is!!!

- Problems of Existing Systems...
  - Single node database (low scalability)
    ArcGIS, PostGIS, Oracle Spatial.
  - Disk-oriented cluster computation (low performance)
    Hadoop-GIS, SpatialHadoop, GeoMesa.
  - No sophisticated query planner and optimizer
    SpatialSpark, GeoSpark
  - No native support for spatial operators
    Spark SQL, MemSQL

Contributions

- Simba: **S**patial **I**n **M**emory **B**ig data **A**nalytics.
  1. Extends Spark SQL to support spatial queries and offers simple APIs for both SQL and DataFrame.
  2. Support two-layer spatial indexing over RDDs (low latency).
  3. Designs a SQL context to run important spatial operations in parallel (high throughput).
  4. Introduces spatial-aware and cost-based optimizations to select good spatial plans.

Contributions

- Simba: **S**patial **I**n **M**emory **B**ig data **A**nalytics.
  1. Extends Spark SQL to support spatial queries and offers simple APIs for both SQL and DataFrame.
  2. Support two-layer spatial indexing over RDDs (low latency).
  3. Designs a SQL context to run important spatial operations in parallel (high throughput).
  4. Introduces spatial-aware and cost-based optimizations to select good spatial plans.

Contributions

- Simba: **S**patial **I**n **M**emory **B**ig data **A**nalytics.
  1. Extends Spark SQL to support spatial queries and offers simple APIs for both SQL and DataFrame.
  2. Support two-layer spatial indexing over RDDs (low latency).
  3. Designs a SQL context to run important spatial operations in parallel (high throughput).
  4. Introduces spatial-aware and cost-based optimizations to select good spatial plans.

Contributions

- Simba: **S**patial **I**n **M**emory **B**ig data **A**nalytics.
  1. Extends Spark SQL to support spatial queries and offers simple APIs for both SQL and DataFrame.
  2. Support two-layer spatial indexing over RDDs (low latency).
  3. Designs a SQL context to run important spatial operations in parallel (high throughput).
  4. Introduces spatial-aware and cost-based optimizations to select good spatial plans.

Outline

Outline

1. Simba Architecture Overview
   - Programming Interface
   - Indexing
   - Spatial Operations
   - Optimization
2. A simple example
3. Conclusions

Spark SQL Overview

Spark SQL is Apache Spark's module for working with structured data.

- Seamlessly mixes SQL queries with Spark programs.
- Connects to any data source the same way.
- Includes a highly extensible cost-based optimizer (*Catalyst*).
- Spark SQL is a full-fledged query engine based on the underlying Spark core.

Spark SQL Overview

Spark SQL is Apache Spark's module for working with structured data.

- Seamlessly mixes SQL queries with Spark programs.
- Connects to any data source the same way.
- Includes a highly extensible cost-based optimizer (*Catalyst*).
- Spark SQL is a full-fledged query engine based on the underlying Spark core.

Spark SQL Overview

```
# Apply functions to results of SQL queries.
context = HiveContext(sc)
results = context.sql("""
                        SELECT
                                *
                        FROM
                                people""")
names = results.map(lambda p: p.name)
# Query and join different data sources.
context.jsonFile("s3n://...").registerTempTable("json")
results = context.sql("""
                        SELECT
                              *
                        FROM
                              people
                        JOIN
                              json ...""")
```

Simba Architecture

Simba is an extension of Spark SQL across the system stack.

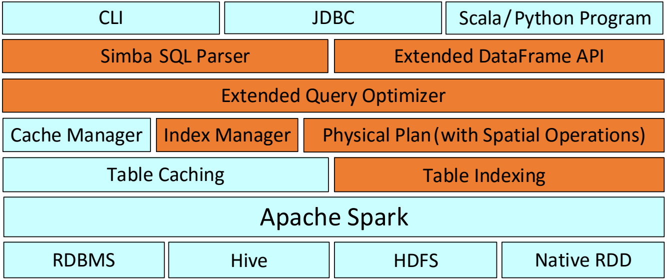


Figure 1: Simba architecture.

Simba Architecture

Simba is an extension of Spark SQL across the system stack.

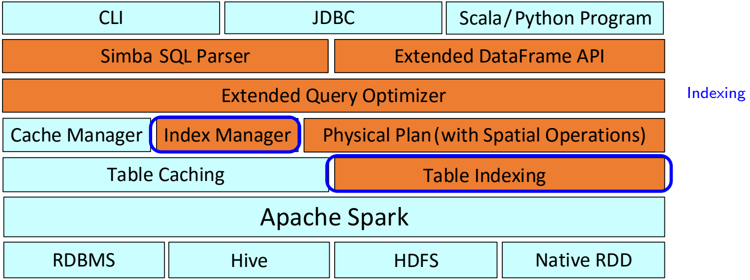


Figure 1: Simba architecture.

Simba Architecture

Simba is an extension of Spark SQL across the system stack.

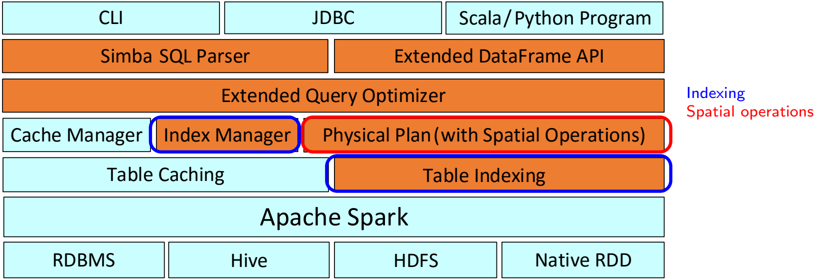


Figure 1: Simba architecture.

Simba Architecture

Simba is an extension of Spark SQL across the system stack.

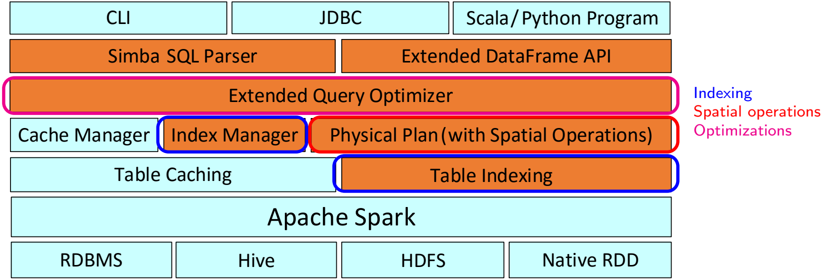


Figure 1: Simba architecture.

Simba Architecture

Simba is an extension of Spark SQL across the system stack[1].

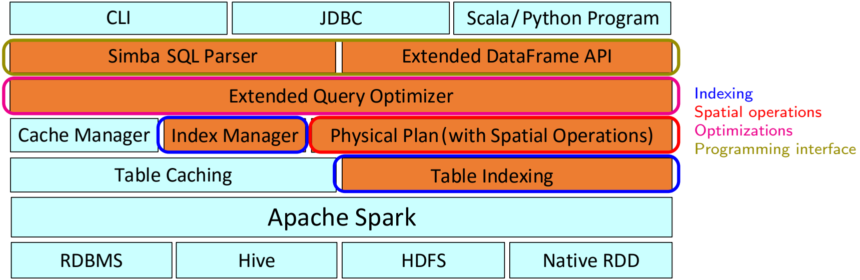


Figure 1: Simba architecture.

[1] a "bit" different in last version over Spark 2.X

Outline

1. Simba Architecture Overview
   - Programming Interface
   - Indexing
   - Spatial Operations
   - Optimization
2. A simple example
3. Conclusions

Programming Interface

- Support rich query types natively in the kernel...
  - The 5 nearest entries to point (2,3).

```
SELECT
    *
FROM
    points
SORT BY
    (x - 2) * (x - 2) +
    (y - 3) * (y - 3)
LIMIT
    5
```

$\Longrightarrow$

```
SELECT
    *
FROM
    points
WHERE
    POINT(x, y) IN
        KNN(POINT(2, 3), 5)
```

Spatial Predicates

- RANGE, CIRCLERANGE and KNN...
  - Show me the points inside a rectangle:

```
SELECT
      *
FROM
      points p
WHERE
      POINT(p.x, p.y) IN RANGE(POINT(10, 5), POINT(15, 8)).
```

  - Show me the points laying 10m around:

```
SELECT
      *
FROM
      points p
WHERE
      POINT(p.x, p.y) IN CIRCLERANGE(POINT(4, 5), 10)
```

  - Show me the 3 nearest points:

```
SELECT
      *
FROM
      points p
WHERE
      POINT(p.x, p.y) IN KNN(POINT(4, 5), 3)
```

Spatial Joins

- KNN JOIN and DISTANCE JOIN...
  - List the 5 nearest hotels around Points of Interest.

```
SELECT
      *
FROM
      hotels AS h
KNN JOIN
      pois AS p
ON
      POINT(p.x, p.y) IN KNN(POINT(h.x, h.y), 5)
```

  - Show me drones that are close to each other (less that 20m).

```
SELECT
      *
FROM
      drones AS d1
DISTANCE JOIN
      drones AS d2
ON
      POINT(d2.x, d2.y, d2.z) IN CIRCLERANGE(POINT(d1.x, d1.y, d1.z), 20.0).
```

Index Management

- `CREATE INDEX` and `DROP INDEX`...
  - Create a 3D index on the sensor table using a R-tree:

```
CREATE INDEX pointIndex ON sensor(x, y, z) USE RTREE

DROP INDEX pointIndex ON sensor
```

  - Generic use:

```
CREATE INDEX idx_name ON R(x1, ..., xm) USE idx_type

DROP INDEX idx_name ON table_name
```

  - Dataset/Dataframe API:

```
dataset.index(RTreeType, "rtDataset",  Array("x", "y"))

dataset.dropIndex()
```

Compound Queries

- Fully compatible with standard SQL operators...
  - Let's count the number of restaurants around 200m of a POI (sort locations by the count):

```
SELECT
      p.id, count(*) AS n
FROM
      pois AS p
DISTANCE JOIN
      restaurants AS r
ON
      POINT(r.lat, r.lng) IN CIRCLERANGE(POINT(p.lat, p.lng), 200.0)
GROUP BY
      p.id
ORDER BY
      n
```

Dataset/DataFrame Support

- Same level of flexibility for Dataset/DataFrames...
  - Let's count the number of restaurants around 200m of a POI (sort locations by the count):

```
pois.distanceJoin(restaurants, Array("pois_lat",
↪  "pois_lon"), Array("rest_lat", "rest_lon"), 200.0)
.groupBy(pois("id"))
.agg(count("*").as("n"))
.sort("n").show()
```

  - Updated examples at https://github.com/InitialDLab/Simba/.../examples

Outline

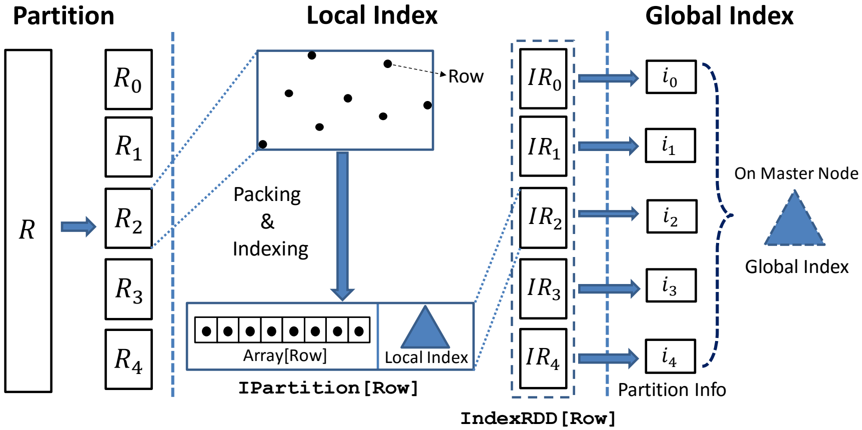
Table Indexing

- In Spark SQL:
  - `Record` $\rightarrow$ `Row`
  - `Table` $\rightarrow$ `RDD[Row]`
- Spark SQL makes a full scan of RDDs.
  - Inefficient for spatial queries!!!
- Solution: native **two-level** indexing over RDDs

Table Indexing

- `IndexRDD`
  - Pack all `Row` objects within a RDD partition into an array (O(1) cost for access).
  - `IPartition` data structure:
    - `case class IPartition[Type](Data: Array[Type], I: Index)`
    - `Index` can be `HashMap`, `TreeMap` or `RTree`.
  - So, by using `Type=Row`:
    - `type IndexRDD[Row] = RDD[IPartition[Row]]`

Two-level indexing strategy



Three-Phases Index Construction

- **Partition**
  - Concerns: Partition size, Data locality and Load balancing.
  - `Partitioner` abstract class.
  - STRPartitioner (based on Sort-Tile-Recursive algorithm) by default[2].
- **Local Index**
  - `RDD[Row]` → `IndexRDD[Row]`.
  - Collects statistics from each partition (number of records, partition boundaries, ...).
- **Global Index**
  - Enables to prune irrelevant partitions.
  - Can use different types of indexes[3] and keep them in memory.

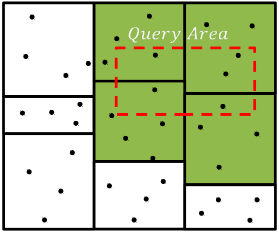
[2] https://github.com/InitialDLab/Simba/.../index

[3] https://github.com/InitialDLab/Simba/.../partitioner

Outline

Range Queries

- $range(Q, R)$
- Two steps: Global filtering + Local processing.




```
SELECT * FROM points p WHERE POINT(p.x, p.y) IN RANGE(POINT(5,5), POINT(10,8))
```

Range Queries

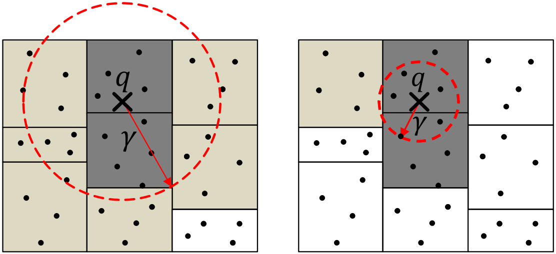
```scala
case class PointData(x: Double, y: Double, z: Double, other: String)

import simba.implicits._
val points = Seq(PointData(1.0, 1.0, 3.0, "1"),
  PointData(2.0, 2.0, 3.0, "2"),
  PointData(2.0, 2.0, 3.0, "3"),
  PointData(2.0, 2.0, 3.0, "4"),
  PointData(3.0, 3.0, 3.0, "5"),
  PointData(4.0, 4.0, 3.0, "6")).toDS()

import simba.simbaImplicits._
points.range(Array("x", "y"),Array(1.0, 1.0),Array(3.0, 3.0)).show(10)
```

kNN Queries

- $kNN(q, R)$
- Good performance thanks to:
  - Local indexes.
  - Pruning bound that is sufficient to cover global kNN results.



(a) Loose Pruning Bound (b) Refined Pruning Bound

```
SELECT * FROM points p WHERE POINT(p.x, p.y) IN KNN(POINT(5,8), 5)
```

kNN Queries

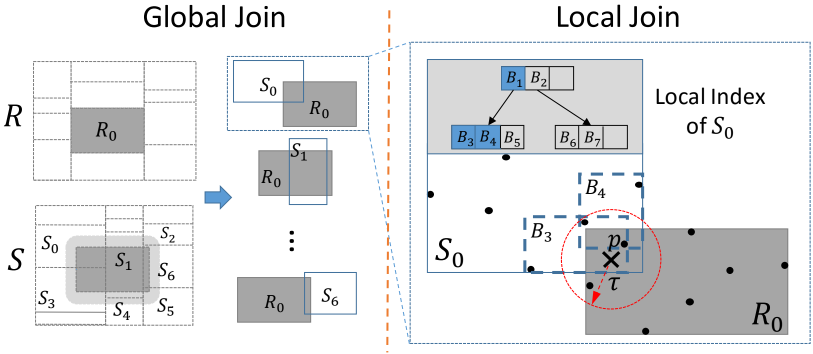
```
case class PointData(x: Double, y: Double, z: Double, other: String)

import simba.implicits._
val points = Seq(PointData(1.0, 1.0, 3.0, "1"),
  PointData(2.0, 2.0, 3.0, "2"),
  PointData(2.0, 2.0, 3.0, "3"),
  PointData(2.0, 2.0, 3.0, "4"),
  PointData(3.0, 3.0, 3.0, "5"),
  PointData(4.0, 4.0, 3.0, "6")).toDS()

import simba.simbaImplicits._
points.knn(Array("x", "y"), Array(1.0, 1.0), 4).show()
```

Distance Join

- $R \bowtie_\tau S$
- DJSpark algorithm.




```
SELECT * FROM R DISTANCE JOIN S ON POINT(S.x, S.y) IN CIRCLERANGE(POINT(R.x, R.y), 5.0)
```

Distance Join

```
case class PointData(x: Double, y: Double, z: Double, other: String)

import simba.implicits._
val DS1 = (0 until 10000)
  .map(x => PointData(x, x + 1, x + 2, x.toString))
  .toDS
val DS2 = (0 until 10000)
  .map(x => PointData(x, x, x + 1, x.toString))
  .toDS

import simba.simbaImplicits._
DS1.distanceJoin(DS2, Array("x", "y"), Array("x", "y"), 3.0).show()
```

kNN Join

- $R \bowtie_{kNN} S$
- General methodology:
  1. Producing buckets: R and S are divided into n_1 (n_2) equal-sized blocks. Every pair of blocks (R_i, S_j) are shuffled to a bucket.
  2. Local kNN join:Performs $kNN(r, S_j)$ for every $r \in R$
  3. Merge: Finds global kNN of every $r \in R$ among its n_2k local $kNNs$.

kNN Join

```scala
case class PointData(x: Double, y: Double, z: Double, other: String)

import simba.implicits._
val DS1 = (0 until 10000)
  .map(x => PointData(x, x + 1, x + 2, x.toString))
  .toDS
val DS2 = (0 until 10000)
  .map(x => PointData(x, x, x + 1, x.toString))
  .toDS

import simba.simbaImplicits._
DS1.knnJoin(DS2, Array("x", "y"), Array("x", "y"), 3).show()
```

Outline

1. Simba Architecture Overview
   - Programming Interface
   - Indexing
   - Spatial Operations
   - Optimization
2. A simple example
3. Conclusions

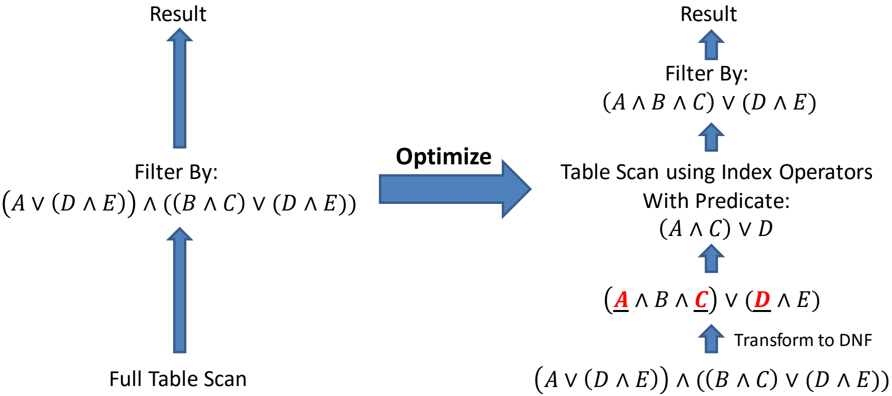
Why does it extend Catalyst?

1. The number of partition plays an important role in performance tuning.
2. Spatial indexes demands new logical optimization rules and spatial predicates management.
3. Indexing optimization cause more overheads than savings (Cost based optimization).

Partition estimation

- Cost model to estimate partition size:
    - Use of a sampling based approach to build estimators.
- Cost model + Partition strategy:
    1. Partitions are balanced.
    2. Each partition fits in memory.
    3. Number of partitions proportional to number of workers.

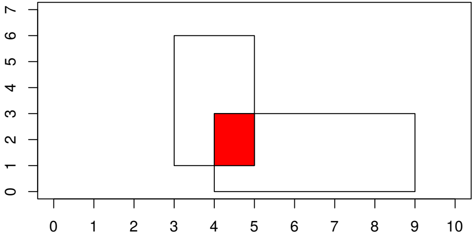
Index awareness optimizations



DNF: Disjunctive Normal Form

Spatial predicates merging

- Geometric properties to merge spatial predicates.
  - i.e. `x > 3 AND x < 5 AND y > 1 AND y < 6` can be merged into a range query on `(POINT(3, 1), POINT(5, 6))`.
  - i.e. Two conjunctive range queries on `(POINT(3, 1), POINT(5, 6)) AND (POINT(4, 0), POINT(9, 3))` can be merged into a single range query on `(POINT(4, 1), POINT(5, 3))`.



Selectivity + CBO

- Selectivity estimation + Cost-based Optimization.
  - Selectivity estimation over local indexes
  - Choose a proper plan: scan or use index.
- Broadcast join optimization: small table joins large table.
- Logical partitioning optimization for kNN joins.
  - Provides tighter pruning bounds.

Outline

A simple example...

```scala
package org.apache.spark.sql.simba.examples

import org.apache.spark.sql.simba.SimbaSession
import org.apache.spark.sql.types.StructType
import org.apache.spark.sql.catalyst.ScalaReflection

object Project {
  case class POI(pid: Long, tags: String, poi_lon: Double, poi_lat: Double)

  def main(args: Array[String]): Unit = {
    val simba = SimbaSession
      .builder()
      .master("local[4]")
      .appName("Project")
      .config("simba.index.partitions", "16")
      .getOrCreate()
```

⋮

A simple example...

⋮

```
    import simba.implicits._
    import simba.simbaImplicits._

    val schema = ScalaReflection.schemaFor[POI].dataType.asInstanceOf[StructType]
    val pois = simba.read.
      option("header", "true").
      schema(schema).
      csv("/home/and/Documents/PhD/TA/CS236FinalProject/Datasets/POIs.csv").
      as[POI]
    pois.show(truncate = false)
    println(s"Number of records: ${pois.count()}")

    simba.stop()
  }
}
```

A simple example...

```
and@and-laptop:~$ spark-submit --class org.apache.spark.sql.simba.examples.Project
↪  /home/and/Documents/PhD/TA/CS236FinalProject/Simba-master/target/scala-2.11/simba_2.11-1.0.jar
+--------+---------------------------------------------------+----------------+----------------+
|pid     |tags                                               |poi_lon         |poi_lat         |
+--------+---------------------------------------------------+----------------+----------------+
|26466687|amenity=pub                                        |-65737.144394621 |3363447.42056986|
|26466690|amenity=pub,name:en=Maya                           |-65480.6907087017|3364134.18568005|
|26466717|amenity=pub                                        |-61189.2883719679|3362555.34129586|
|26484067|amenity=parking                                    |-65748.3494233086|3359156.83818014|
|26488397|amenity=cafe,name:en=Starbucks                     |-64758.6595641028|3364843.62027897|
|26607397|amenity=restaurant                                 |-63449.5236124868|3361023.47129964|
|26882571|amenity=fuel,name:en=Sinopec                       |-64517.7142340408|3367488.92447102|
|26932786|amenity=fuel,name:en=Sinopec                       |-60843.6688328821|3365846.16210334|
|27117771|amenity=pub                                        |-65638.2572549942|3363934.59573984|
|27181039|amenity=cafe,name:en=Starbucks                     |-62686.3406153971|3362704.11640486|
|27181040|amenity=cafe,name:en=Starbucks                     |-62773.2588839596|3363103.01586046|
|27246222|amenity=restaurant,name:en=New White Deer Restaurant |-62984.0964777985|3363947.50760504|
|27262500|amenity=fast_food,name:en=KFC                      |-62543.1297553628|3363095.07490381|
|27262504|amenity=fast_food,name:en=KFC                      |-62620.7551165471|3363061.87760935|
|27262513|amenity=fast_food,name:en=KFC                      |-66170.913779046 |3365787.87254387|
|27262517|amenity=restaurant,name:en=Chuan Wei Guan          |-61980.578202843 |3363451.94953999|
|27446997|amenity=bus_station,name:en=Hangzhou West Bus Station|-69565.5633025697|3364138.23749985|
+--------+---------------------------------------------------+----------------+----------------+
only showing top 20 rows

Number of records: 61660
```

Outline

1. Simba Architecture Overview
   - Programming Interface
   - Indexing
   - Spatial Operations
   - Optimization
2. A simple example
3. Conclusions

Conclusions

- Simba: A distributed in-memory spatial analytics engine.
- Indexing support for efficient query processing.
- Spatial operator implementation tailored towards Spark.
- Spatial and index-aware optimizations.
- User-friendly SQL and DataFrame API.
- Superior performance compared against other systems.

Thank you!!!

Do you have any question?