

Simba: Efficient In-Memory Spatial Analytics.

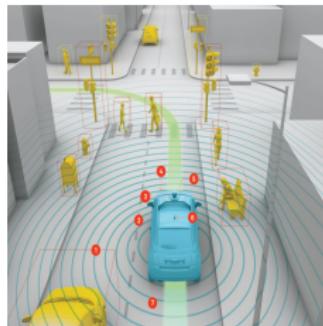
Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou and Minyi Guo
SIGMOD'16.

Andres Calderon

November 10, 2016

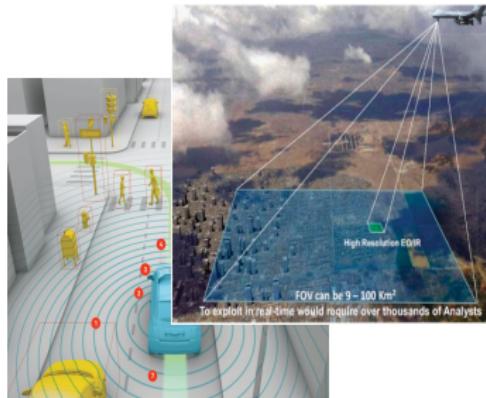
Introduction

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



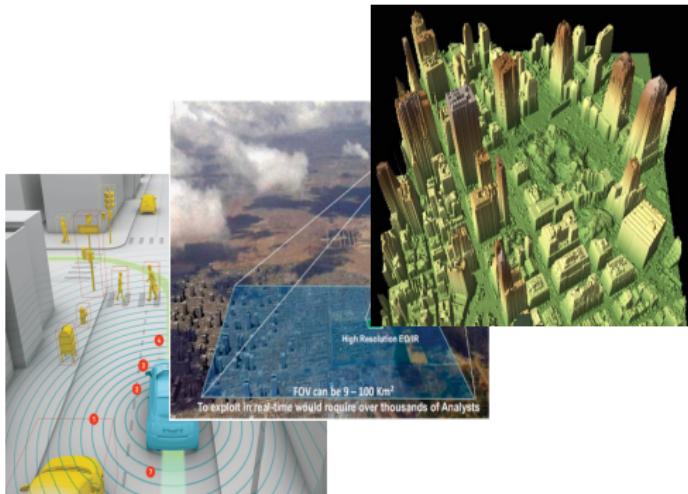
Introduction

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



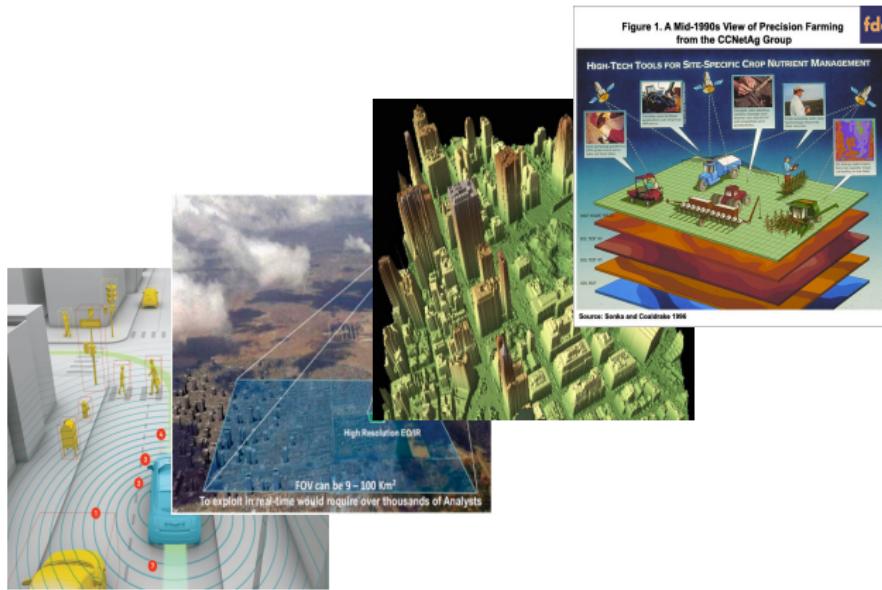
Introduction

- 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...



ZigBee®



waze



UBER



Google

foursquare®

Spatial is Special

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



Spatial is Special

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



MD-Hbase



Spatial is Special

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



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: **Spatial In Memory Big data Analytics.**
 - 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: **Spatial In Memory Big data Analytics.**
 - 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: **Spatial In Memory Big data Analytics.**
 - 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: **Spatial In Memory Big data Analytics.**
 - 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

1 Simba Architecture Overview

- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

Outline

1 Simba Architecture Overview

- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

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.

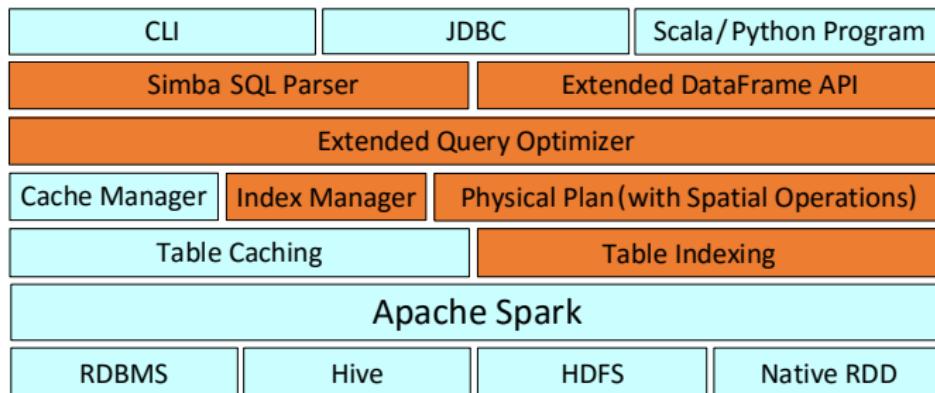


Figure 1: Simba architecture.

Simba Architecture

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

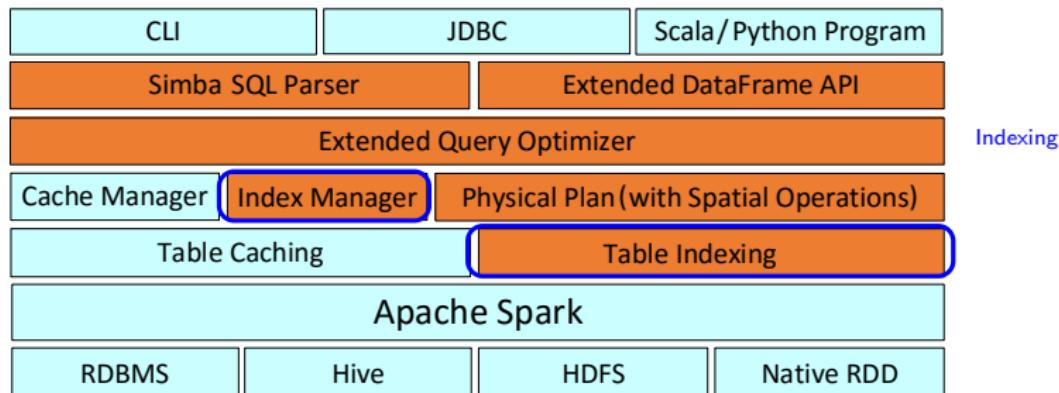


Figure 1: Simba architecture.

Simba Architecture

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

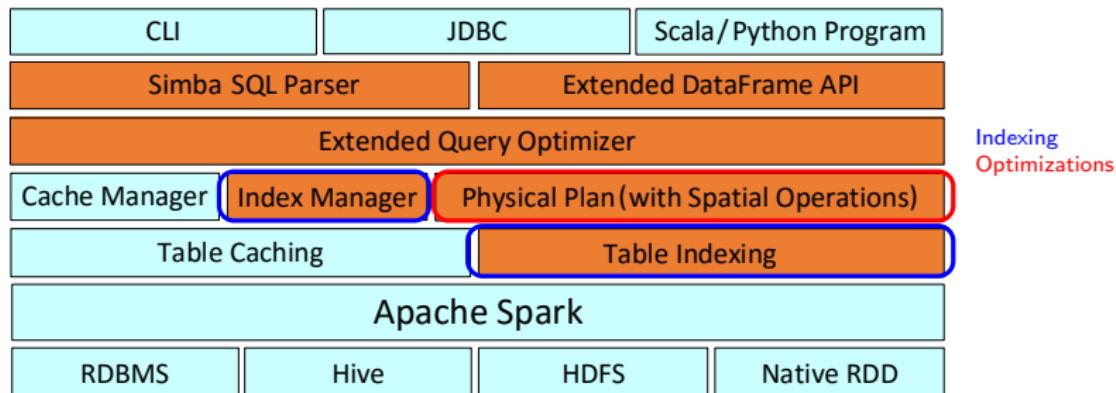


Figure 1: Simba architecture.

Simba Architecture

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

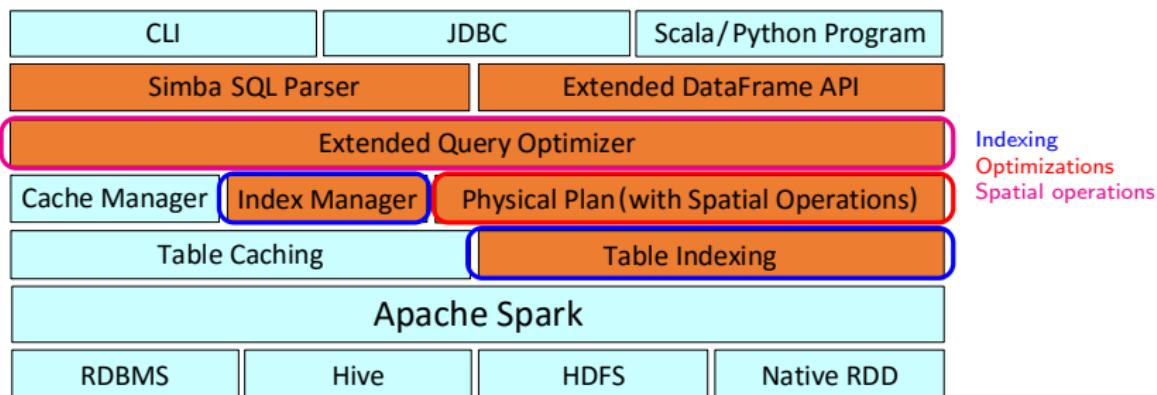


Figure 1: Simba architecture.

Simba Architecture

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

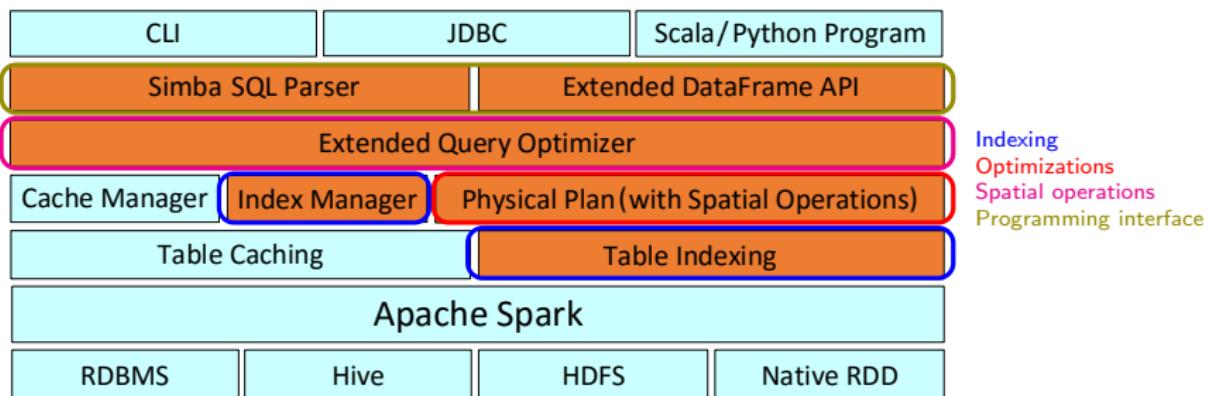


Figure 1: Simba architecture.

Outline

1 Simba Architecture Overview

- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

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
```



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

Spatial Predicates

- RANGE, CIRCLE RANGE 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 CIRCLE RANGE(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 than 20m).

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

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 CIRCLE RANGE(POINT(p.lat, p.lng), 200.0)
GROUP BY
    p.id
ORDER BY
    n
```

DataFrame Support

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

```
pois.distanceJoin(restaurants, Point(pois("lat"),  
    ↪ pois("lng")), Point(restaurants("lat"),  
    ↪ restaurants("lng")), 200.0)  
.groupBy(pois("id"))  
.agg(count("*").as("n"))  
.sort("n").show()
```

Outline

1 Simba Architecture Overview

- Programming Interface
- **Indexing**
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

Table Indexing

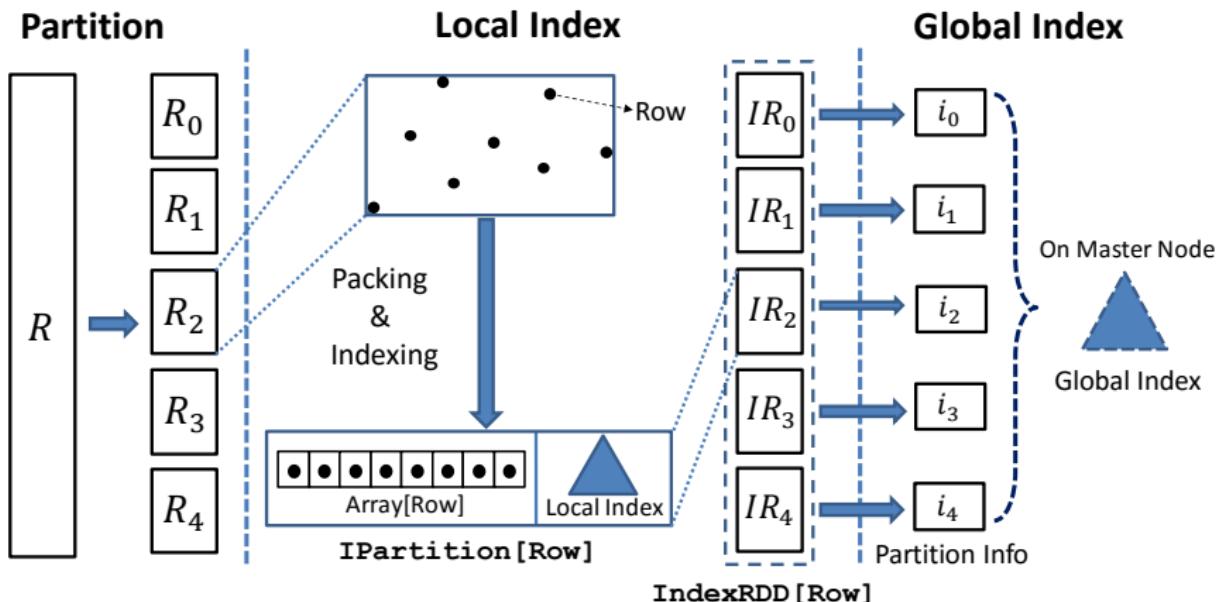
- In Spark SQL:
 - Record → Row
 - Table → RDD [Row]
- Spark SQL makes a full scan of RDDs.
 - Inefficient for spatial queries!!!
- Solution: native **two-level** indexing over RDDs

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).

- **Local Index**

- $\text{RDD}[\text{Row}] \rightarrow \text{IndexRDD}[\text{Row}]$.
- Collects statistics from each partition (number of records, partition boundaries, ...).

- **Global Index**

- Enables to prune irrelevant partitions.
- Can use different types of indexes and keep them in memory.

Outline

1 Simba Architecture Overview

- Programming Interface
- Indexing
- **Spatial Operations**
- Optimization

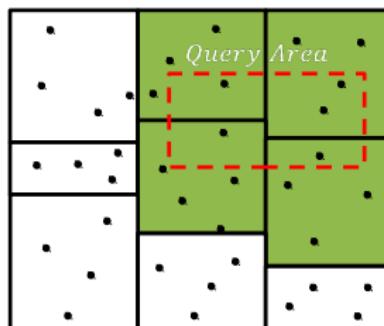
2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

Range Queries

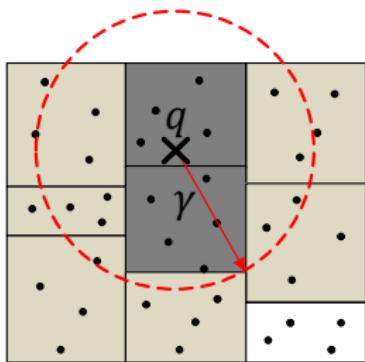
- $\text{range}(Q, R)$
- Two steps: Global filtering + Local processing.



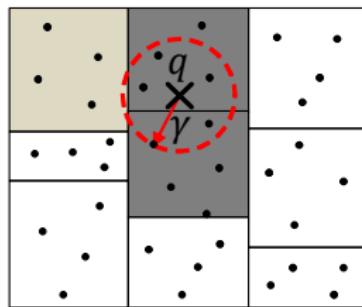
```
SELECT * FROM points p WHERE POINT(p.x, p.y) IN RANGE(POINT(5,5), POINT(10,8))
```

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

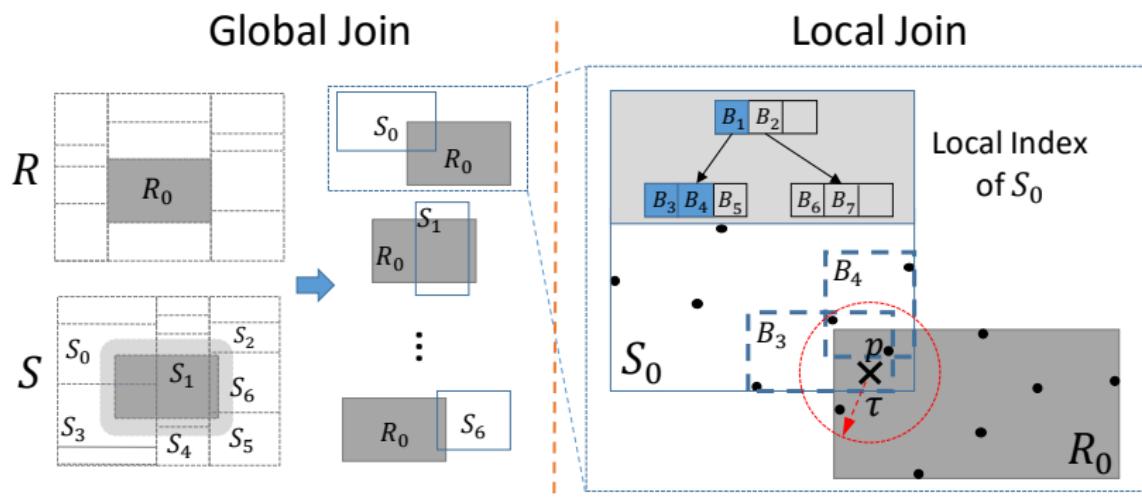


(b) Refined Pruning Bound

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

Distance Join

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



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

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 $n_2 k$ local $kNNs$.

kNN Join

- $R \bowtie_{kNN} S$
- Explores several methods:
 - BKJSpark-N: Block nested loop kNN join in Spark.
 - BKJSpark-R: Block R-tree kNN join in Spark.
 - VKJSpark: Voronoi kNN join in Spark.
 - ZKJSpark: z-value kNN join in Spark.
 - RKJSpark: R-tree kNN join in Spark.

Outline

1 Simba Architecture Overview

- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

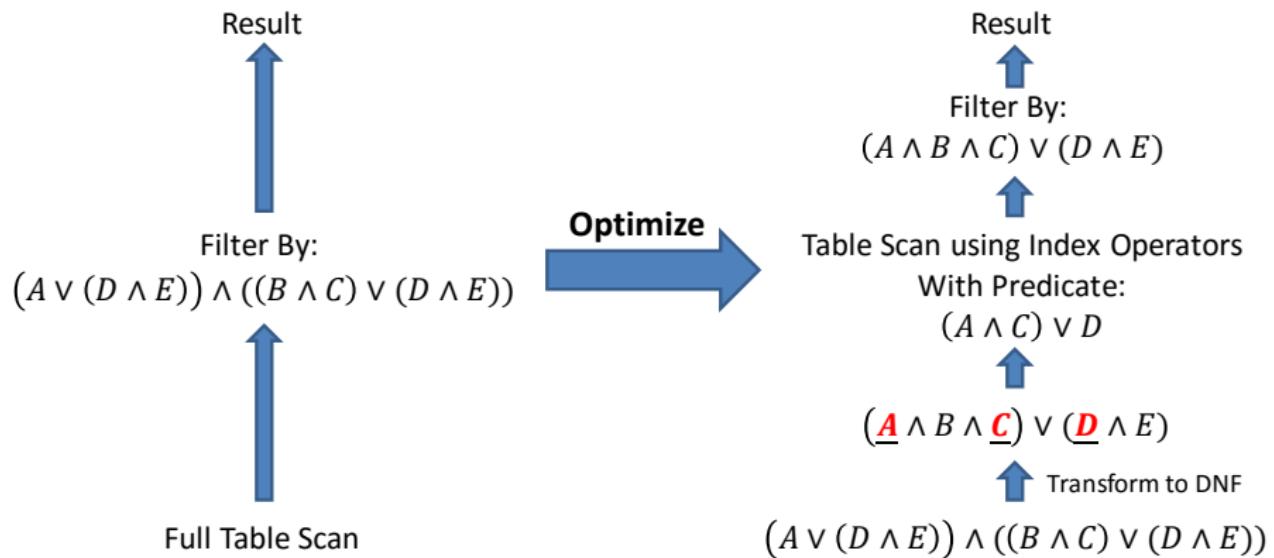
Why does it extend Catalyst?

- 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

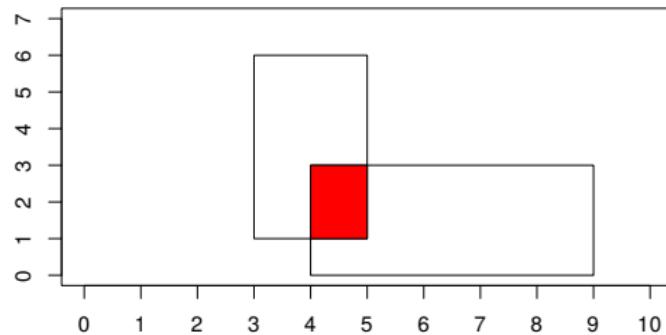
- Cost model to estimate partition size:
 - Use of a sampling based approach to build estimators.
- Cost model + Partition strategy:
 - ① Partitions are balanced.
 - ② Each partition fits in memory.
 - ③ Number of partitions proportional to number of workers.

Index awareness optimizations



Spatial predicates merging

- Geometric properties to merge spatial predicates.
 - i.e. $x > 3 \text{ AND } x < 5 \text{ AND } y > 1 \text{ AND } y < 6$ 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

1 Simba Architecture Overview

- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

Setup

- 10 nodes cluster
- Processors: 6-core Intel Xeon E5 (1.6 to 2.0 GHz)
- RAM: 20 to 56 GB.
- Ubuntu 14.04 LTS, Hadoop 2.4.1, Spark 1.3.0

Datasets

- OSM (OpenStreetMap)
 - 2.2 Billion records, 132GB.
 - Five fields: ID, a two-dimensional coordinate and two text information.
- GDEL (Global Data on Events, Language and Tone)
 - 75 Million records
 - Seven attributes: timestamp, three two-dimensional coordinates (start, end and action of the event).
- RC (Synthetic dataset)
 - 1 Million to 1 Billion records, 2 to 6 dimensions.
 - Clusters randomly generated using Gaussian distributions.

Outline

1 Simba Architecture Overview

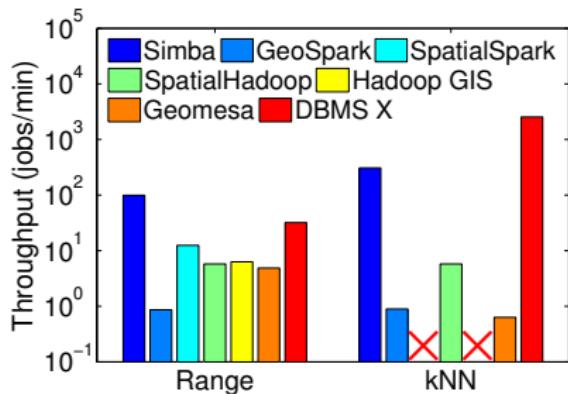
- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

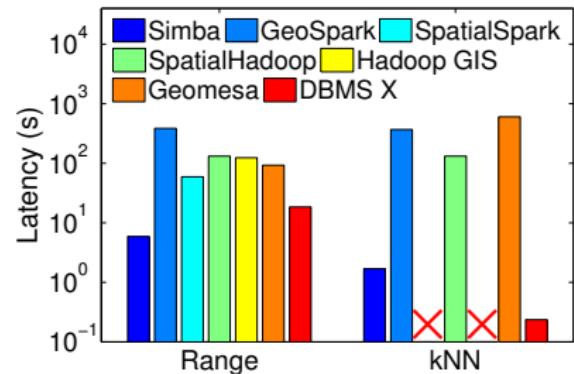
- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

Range and kNN Operations (OSM)

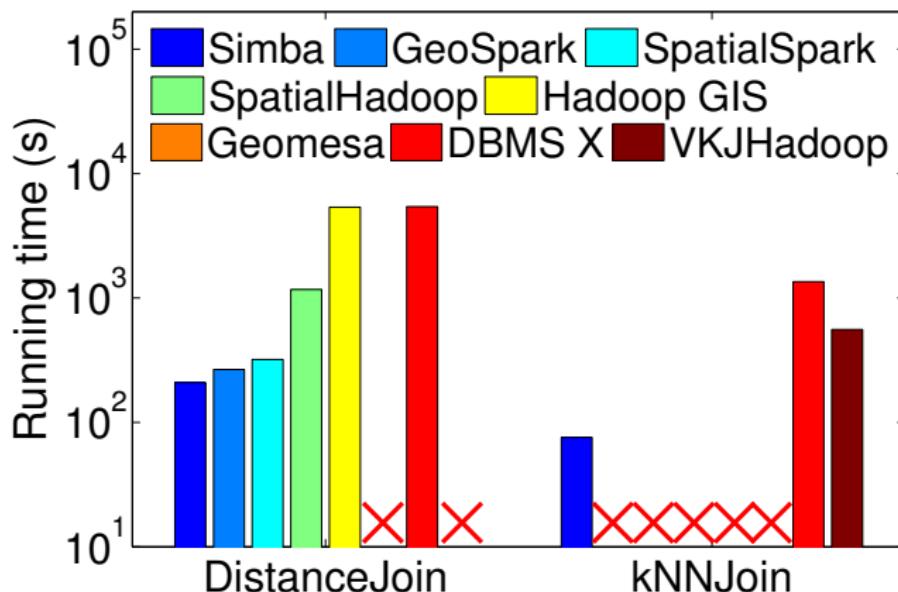


(a) Throughput

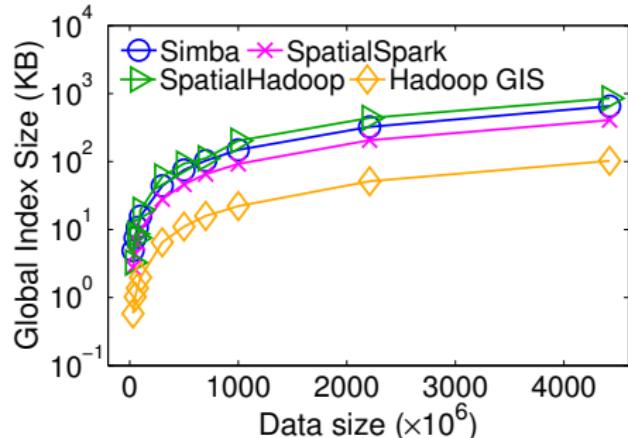
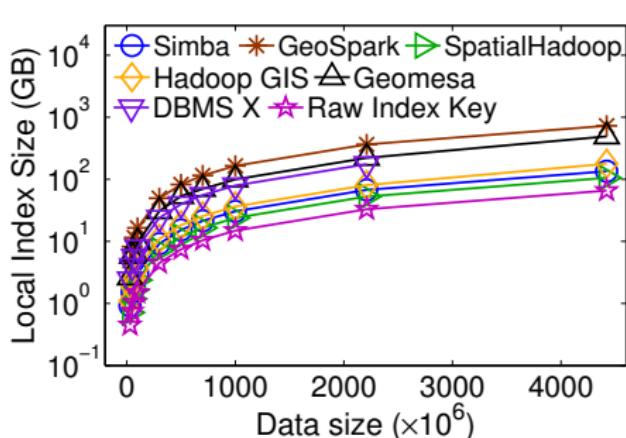


(b) Latency

Join Operations (OSM)



Index Storage Overhead (OSM)



Outline

1 Simba Architecture Overview

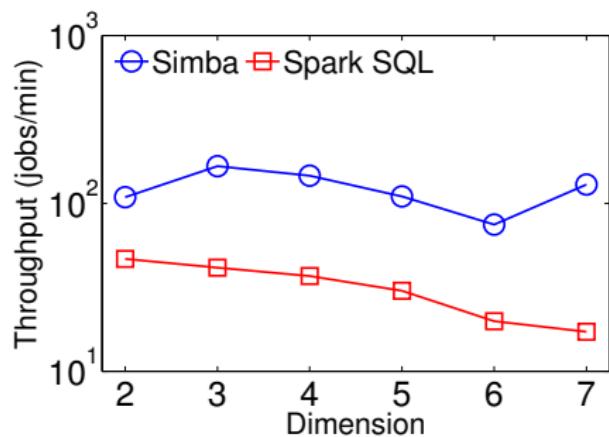
- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

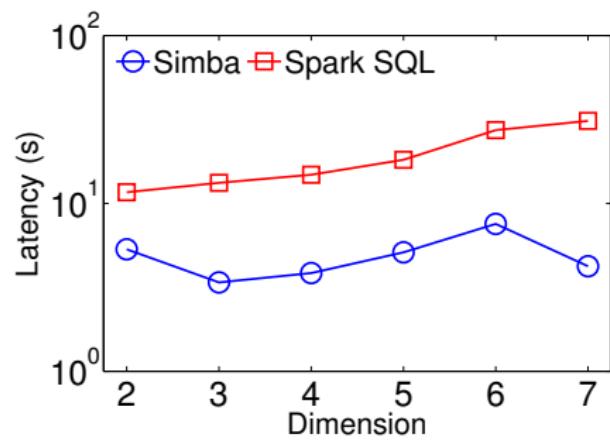
- Comparison with Existing Systems
- **Comparison against Spark SQL**
- Join Methods vs Dimensionality

3 Conclusions

Range Query Performance (GDELT)

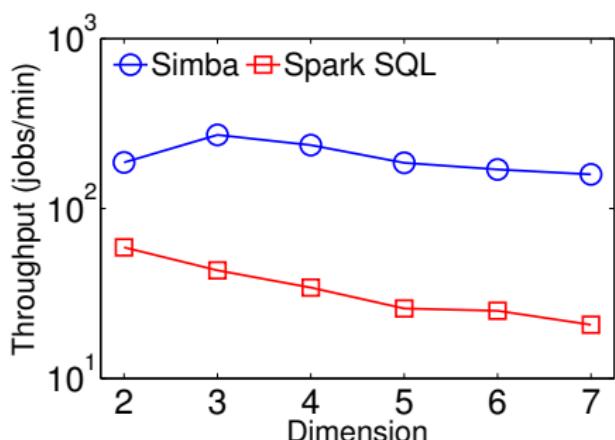


(a) Throughput

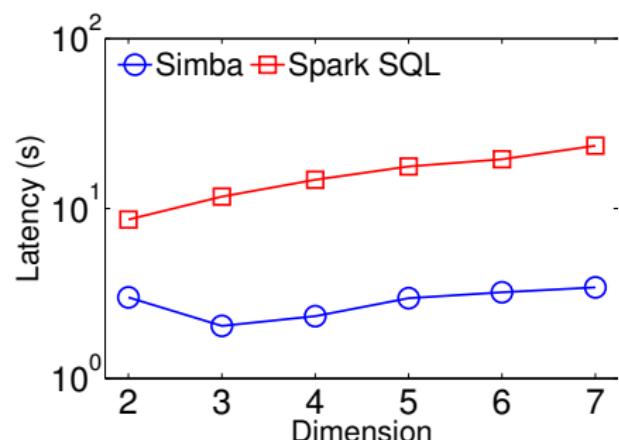


(b) Latency

kNN Query Performance (GDELT)



(a) Throughput



(b) Latency

Outline

1 Simba Architecture Overview

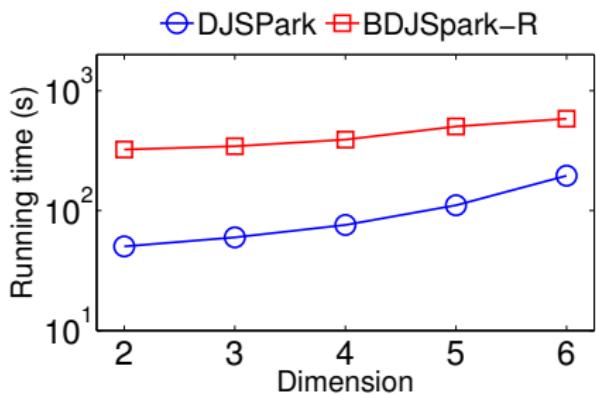
- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

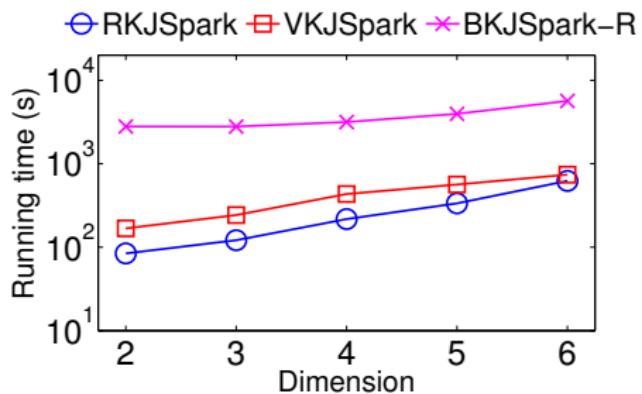
- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

3 Conclusions

Join Operations Performance (RC)



(a) Distance join



(b) k NN join

Outline

1 Simba Architecture Overview

- Programming Interface
- Indexing
- Spatial Operations
- Optimization

2 Experiments

- Comparison with Existing Systems
- Comparison against Spark SQL
- Join Methods vs Dimensionality

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.
 - No changes to Spark kernel, easier migration to higher version Spark.
 - Superior performance compared against other systems.

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.
- No changes to Spark kernel, easier migration to higher version Spark.
- Superior performance compared against other systems.

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.
- No changes to Spark kernel, easier migration to higher version Spark.
- Superior performance compared against other systems.

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.
- No changes to Spark kernel, easier migration to higher version Spark.
- Superior performance compared against other systems.

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.
- No changes to Spark kernel, easier migration to higher version Spark.
- Superior performance compared against other systems.

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.
- No changes to Spark kernel, easier migration to higher version Spark.
- Superior performance compared against other systems.

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.
- No changes to Spark kernel, easier migration to higher version Spark.
- Superior performance compared against other systems.

Future ideas

- Native support to geometry objects and operations.
- Spatial joins over predicates (i.e. intersect or touch).
- Explore complex spatio-temporal patterns.

Future ideas

- Native support to geometry objects and operations.
- Spatial joins over predicates (i.e. intersect or touch).
- Explore complex spatio-temporal patterns.

Future ideas

- Native support to geometry objects and operations.
- Spatial joins over predicates (i.e. intersect or touch).
- Explore complex spatio-temporal patterns.

Future ideas

- Native support to geometry objects and operations.
- Spatial joins over predicates (i.e. intersect or touch).
- Explore complex spatio-temporal patterns.

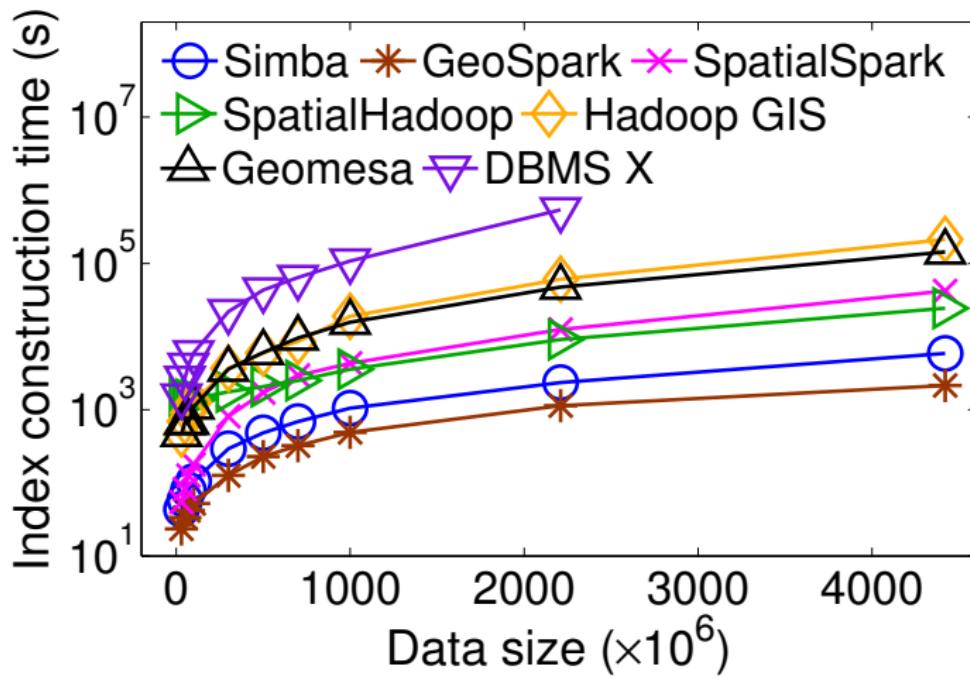
Thank you!!!

Do you have any question?

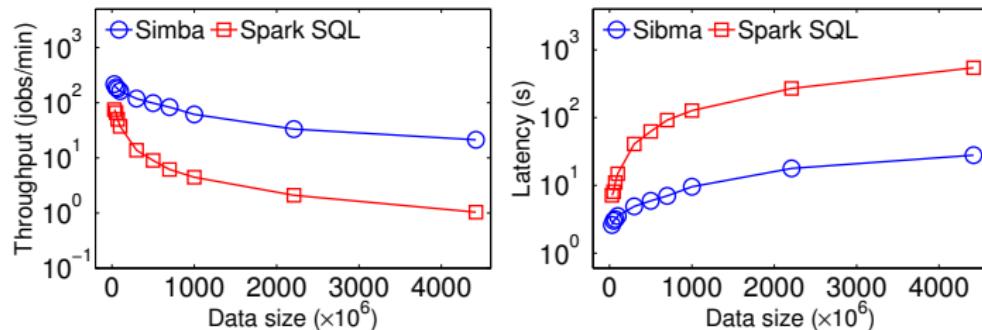
Comparison with existing systems

Core Features	Simba	GeoSpark	SpatialSpark	SpatialHadoop	Hadoop GIS
Data dimensions	multiple	$d \leq 2$	$d \leq 2$	$d \leq 2$	$d \leq 2$
SQL	✓	✗	✗	Pigeon	✗
DataFrame API	✓	✗	✗	✗	✗
Spatial indexing	R-tree	R-/quad-tree	grid/kd-tree	grid/R-tree	SATO
In-memory	✓	✓	✓	✗	✗
Query planner	✓	✗	✗	✓	✗
Query optimizer	✓	✗	✗	✗	✗
Concurrent query execution	thread pool in query engine	user-level process	user-level process	user-level process	user-level process
query operation support					
Box range query	✓	✓	✓	✓	✓
Circle range query	✓	✓	✓	✗	✗
k nearest neighbor	✓	✓	only 1NN	✓	✗
Distance join	✓	✓	✓	via spatial join	✓
k NN join	✓	✗	✗	✗	✗
Geometric object	✗ ¹	✓	✓	✓	✓
Compound query	✓	✗	✗	✓	✗

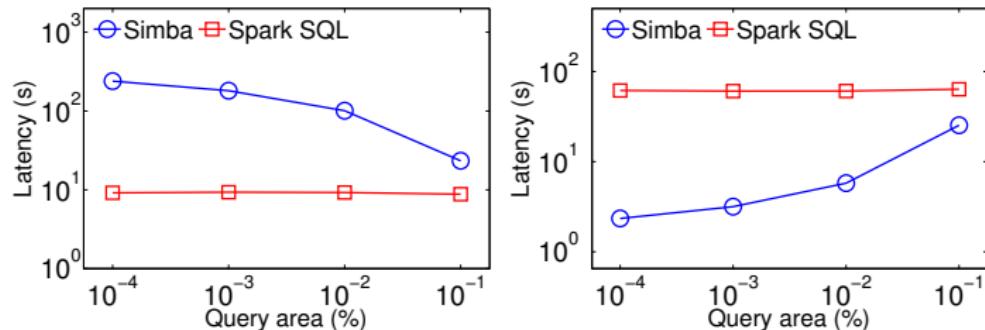
Index Construction Time (OSM)



Range Query Performance (OSM)

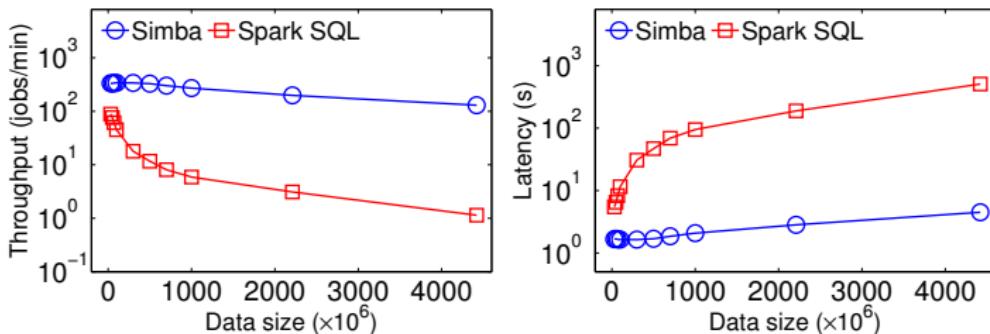


(a) Effect of data size.

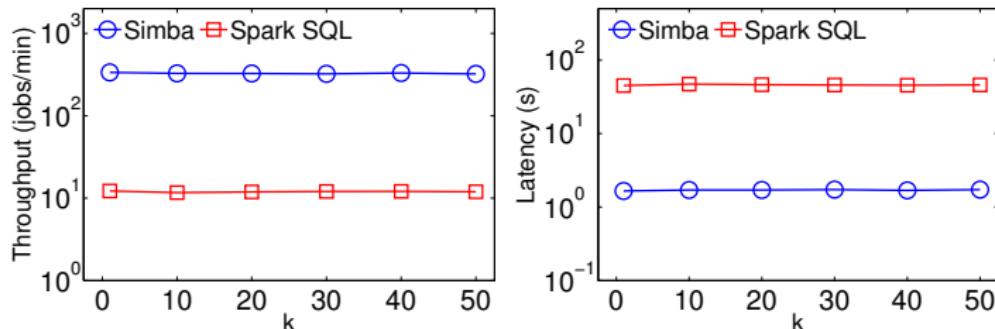


(b) Effect of query area size (percentage of total area).

kNN Query Performance (OSM)

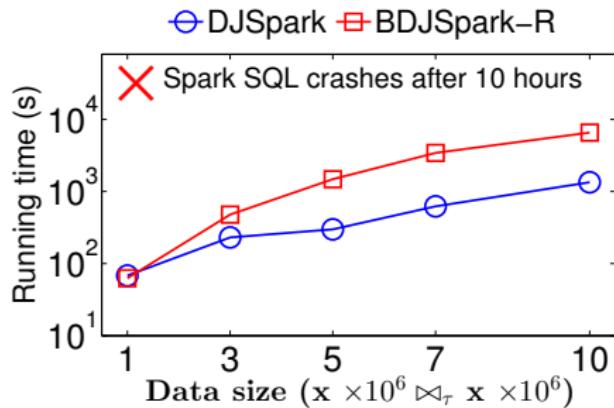


(a) Effect of data size.

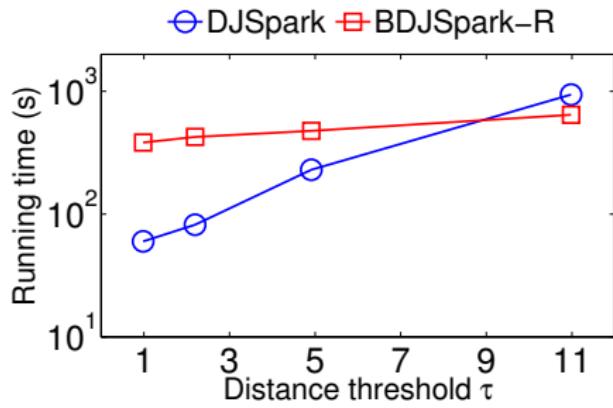


(b) Effect of k .

Distance Join Performance (OSM)

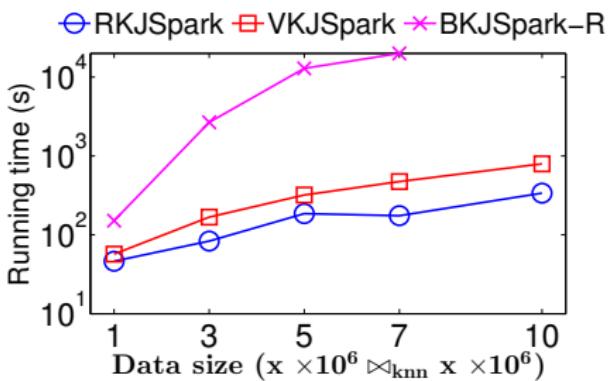


(a) Effect of data size.

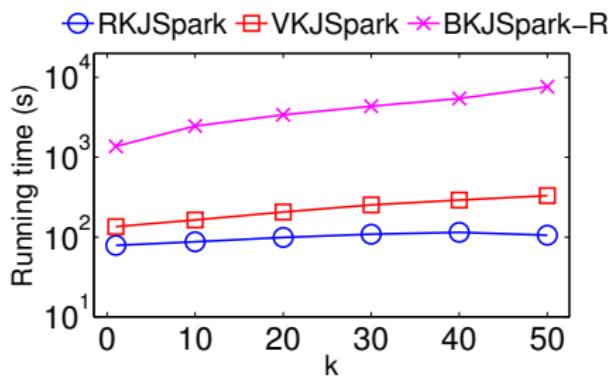


(b) Effect of τ .

kNN Join Performance (OSM)



(a) Effect of data size.



(b) Effect of k .