Simba: Efficient In-Memory Spatial Analytics.

Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou and Minyi Guo

SIGMOD'16.

Andres Calderon

November 9, 2016

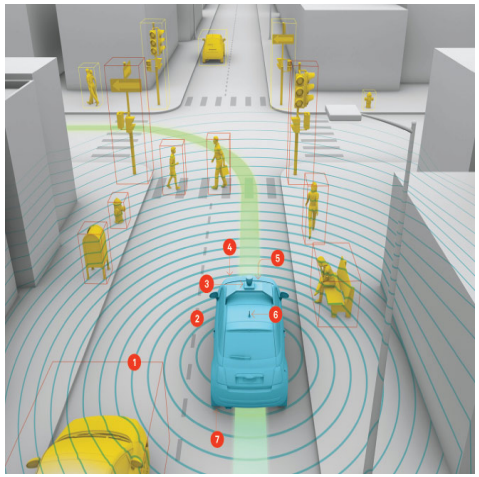
Agenda

1. Background
2. Simba Architecture Overview
   - Programming Interface
   - Indexing
   - Spatial Operations
   - Optimization
3. Experiments
   - Comparison with Existing Systems
   - Comparison against Spark SQL
   - Distance and kNN Join Methods
4. Conclusions

Agenda

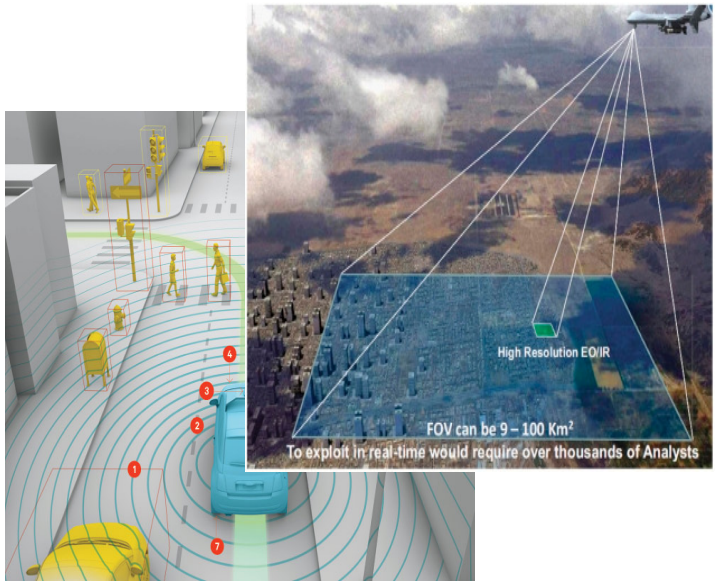
Introduction

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



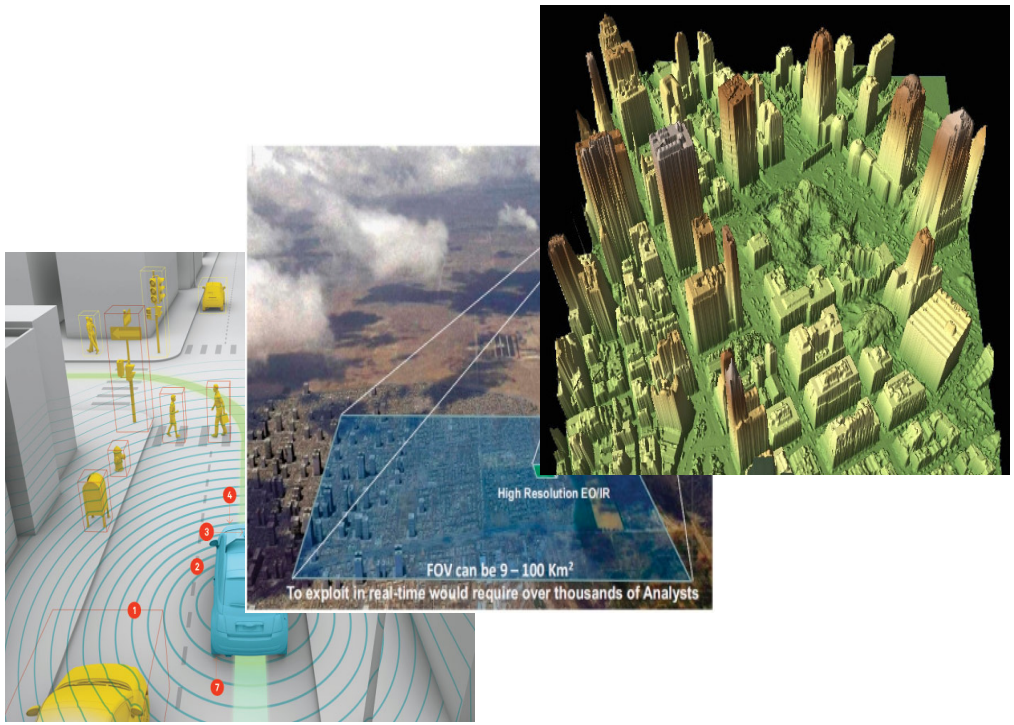
Introduction

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



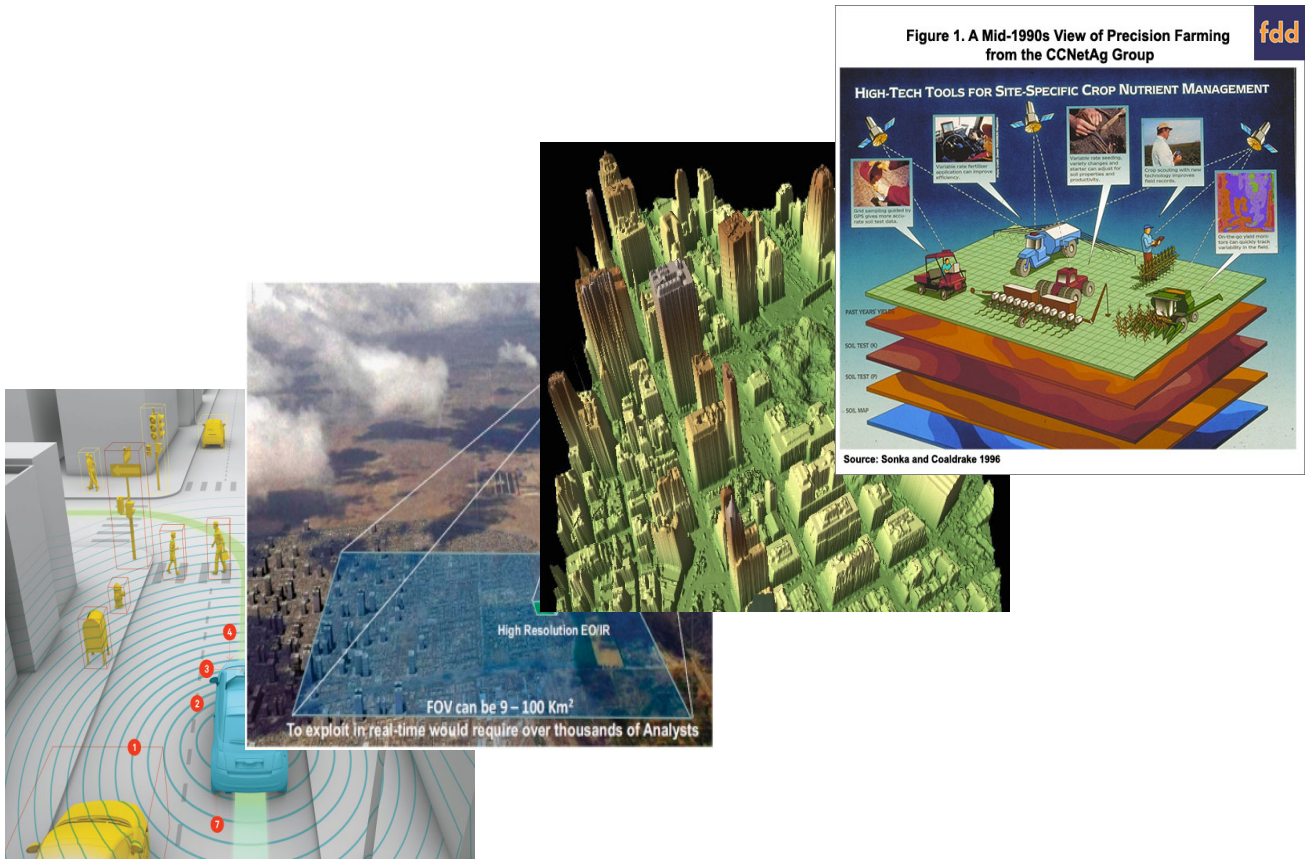
Introduction

- 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...



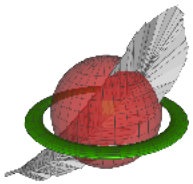
Introduction

- The applications and commercial interest is clear...

Introduction

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

ORACLE SPATIAL



MySQL

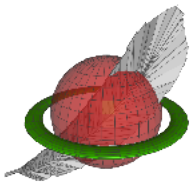
ArcGIS
ESRI

Microsoft SQL Server

Introduction

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

ORACLE SPATIAL



MySQL

ArcGIS
ESRI

Microsoft SQL Server

Hadoop-GIS
Spatial Big Data Solutions

MD-Hbase

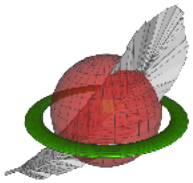
SECONDO

geomesa

Introduction

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

ORACLE SPATIAL



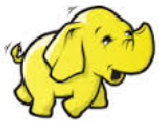
MySQL

ArcGIS
ESRI

Microsoft SQL Server

Hadoop-GIS
Spatial Big Data Solutions

MD-Hbase



SECONDO

geomesa

GeoSpark

GeoTrellis

Spark SQL

SpatialSpark

MEMSQL

Introduction

- Why do we need a new tool???

Introduction

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

Introduction

- 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.

Introduction

- 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.

Introduction

- 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.

Introduction

- 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.

Introduction

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	✓
kNN join	✓	×	×	×	×
Geometric object	×[1]	✓	✓	✓	✓
Compound query	✓	×	×	✓	×

Agenda

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.

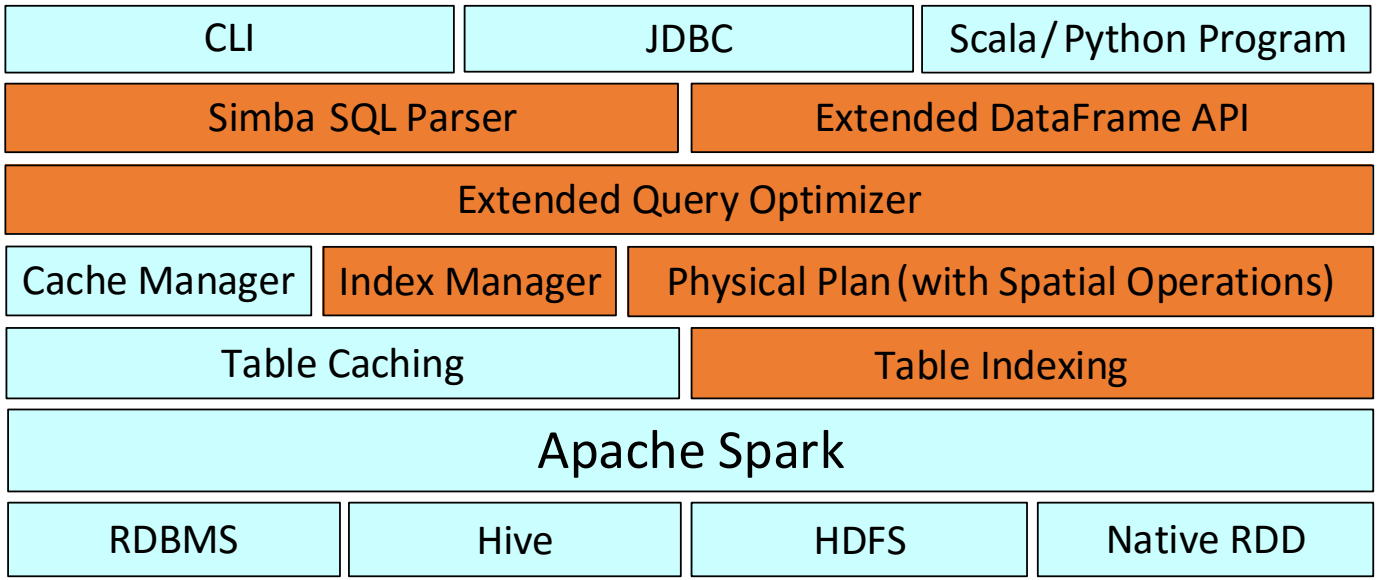


Figure 1: Simba architecture.

Simba Architecture

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

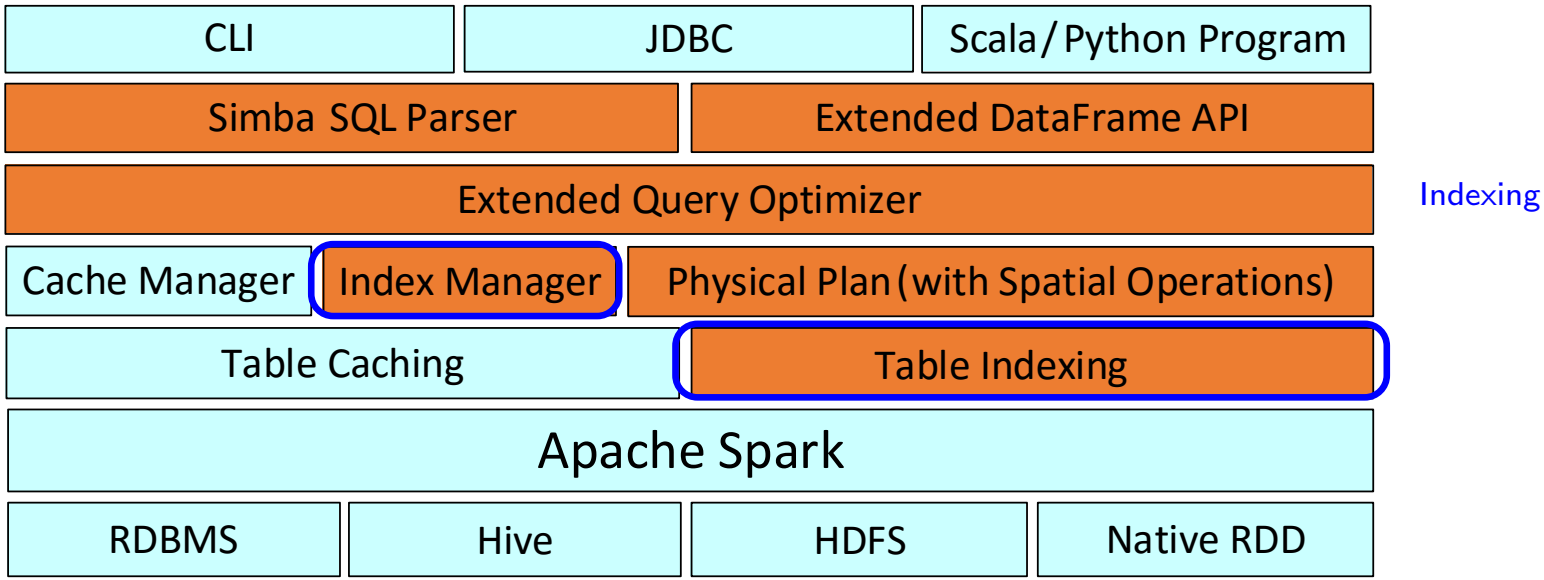


Figure 1: Simba architecture.

Simba Architecture

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

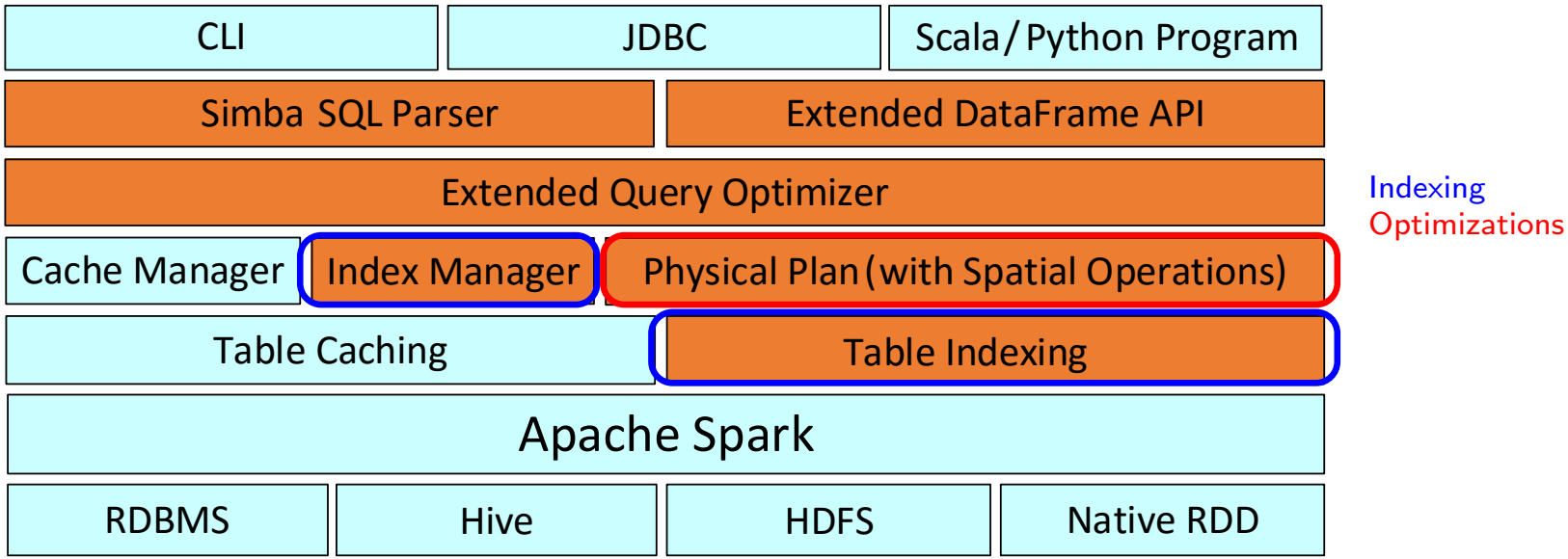


Figure 1: Simba architecture.

Simba Architecture

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

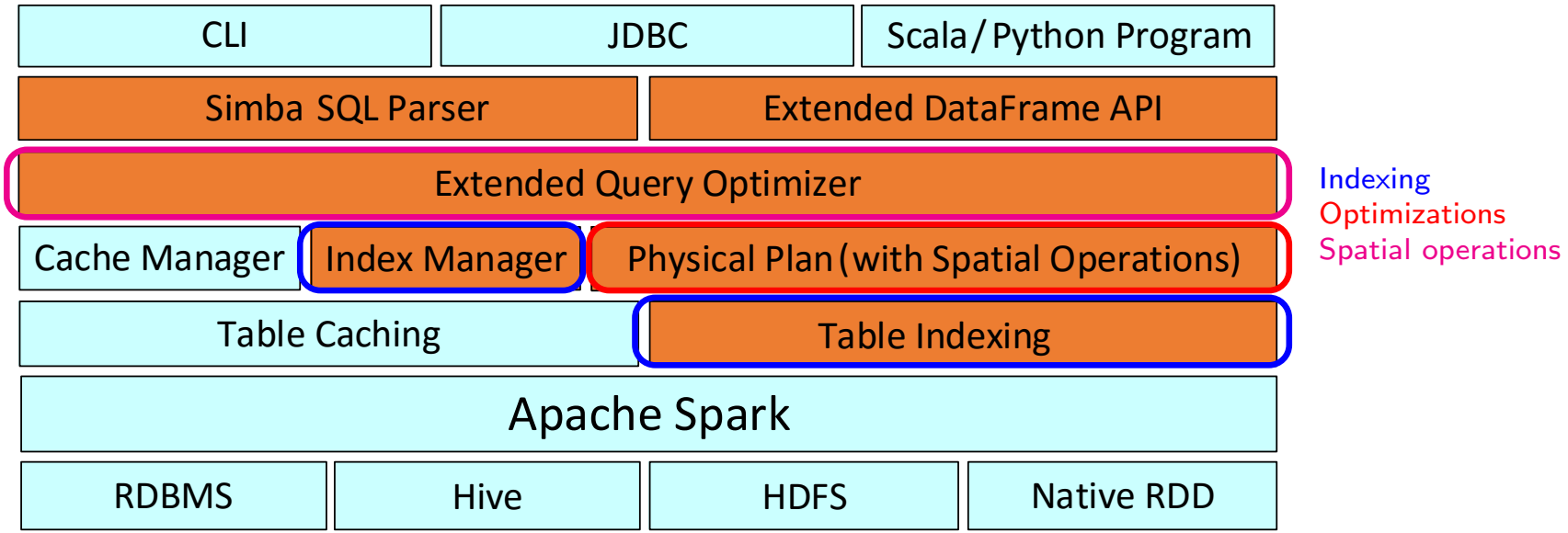


Figure 1: Simba architecture.

Simba Architecture

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

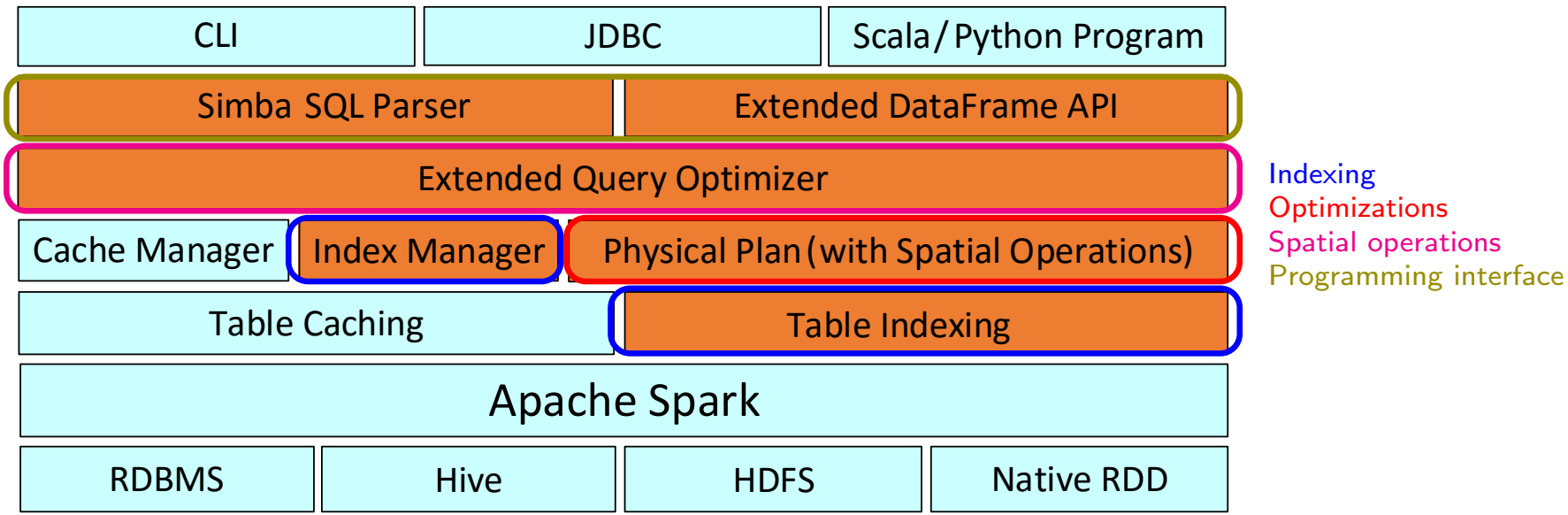
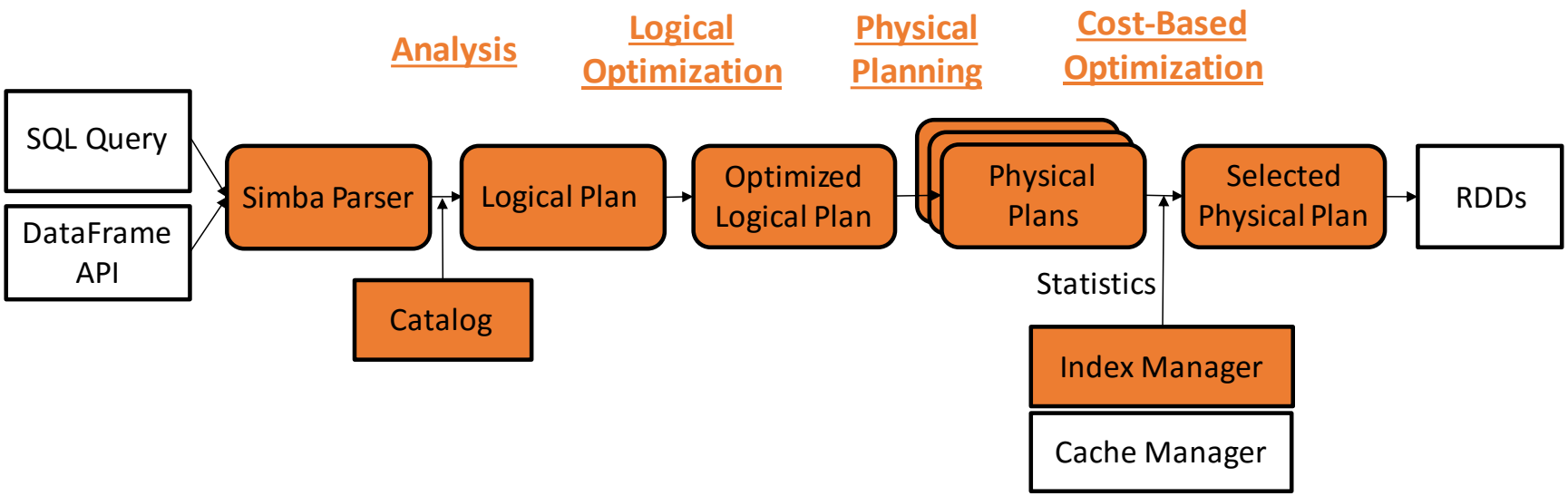


Figure 1: Simba architecture.

Query Load in Simba



Agenda

Programming Interface

Support rich query types natively in the kernel...

```
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...

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

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

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

Spatial Joins

DISTANCE JOIN and KNN JOIN...

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

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

```
CREATE INDEX idx_name ON R(x_1, ..., x_m) USE idx_type
DROP INDEX idx_name ON table_name
```

Compound Queries

Fully compatible with standard SQL operators...

```
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
```

DataFrame Support

Same level of flexibility for DataFrames...

```
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()
```

Agenda

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]]`

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**
  - `RDD[Row]` $\rightarrow$ `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 and keep them in memory.

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**
  - `RDD[Row]` $\rightarrow$ `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 and keep them in memory.

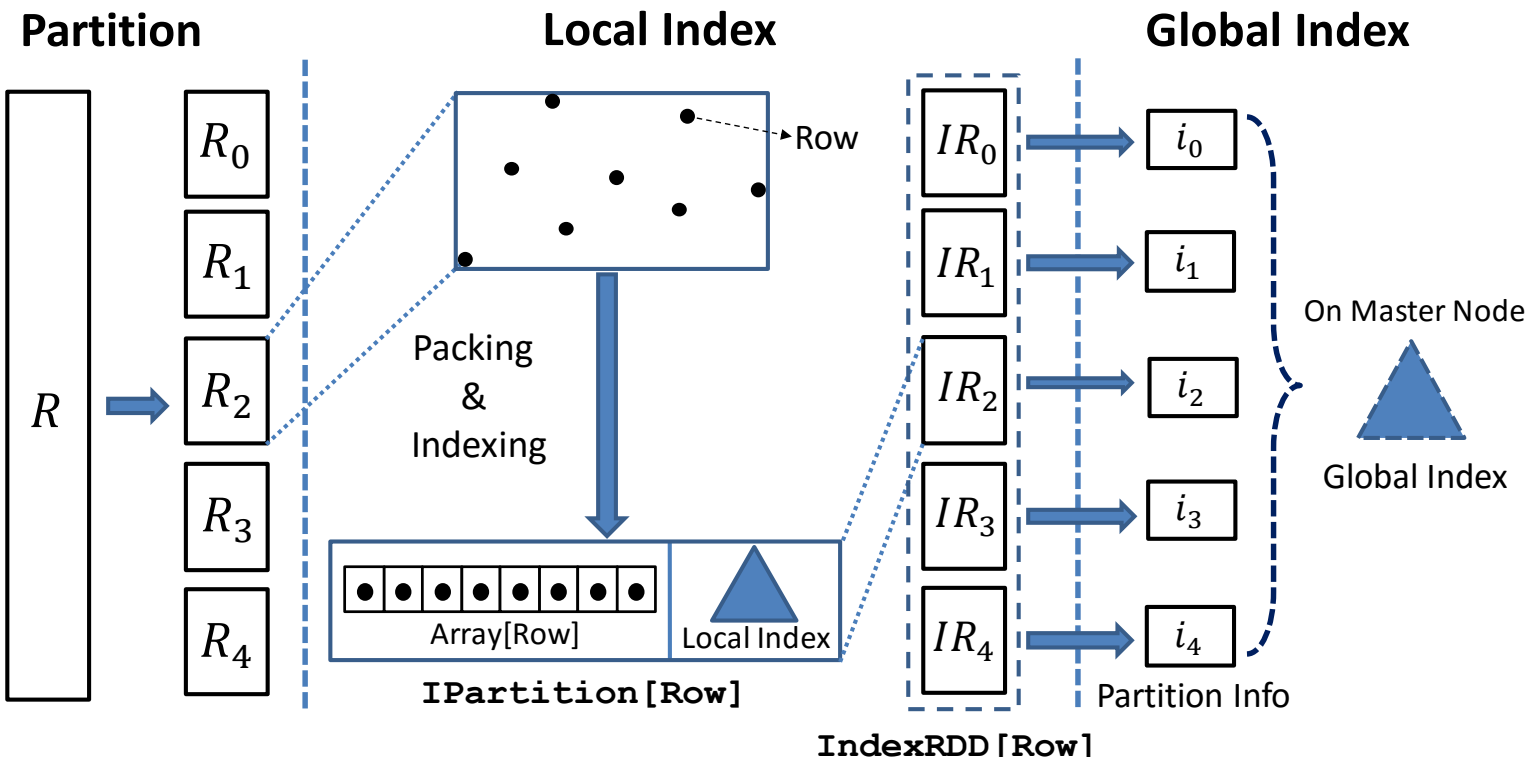
Three-Phases Index Construction

- **Partition**
  - Concerns: Partition size, Data locality and Load balancing.
  - `Partitioner` abstract class.
  - STRPartitioner (based on Sort-Tile-Recursive algorithm).
- **Local Index**
  - `RDD[Row]` $\rightarrow$ `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 and keep them in memory.

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**
  - `RDD[Row]` $\rightarrow$ `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 and keep them in memory.

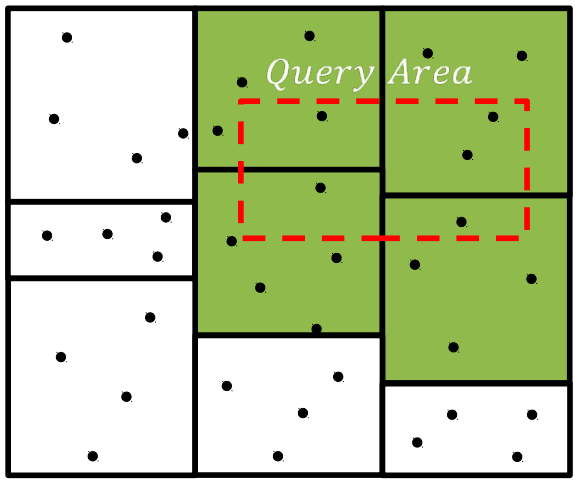
Two-level indexing strategy



Agenda

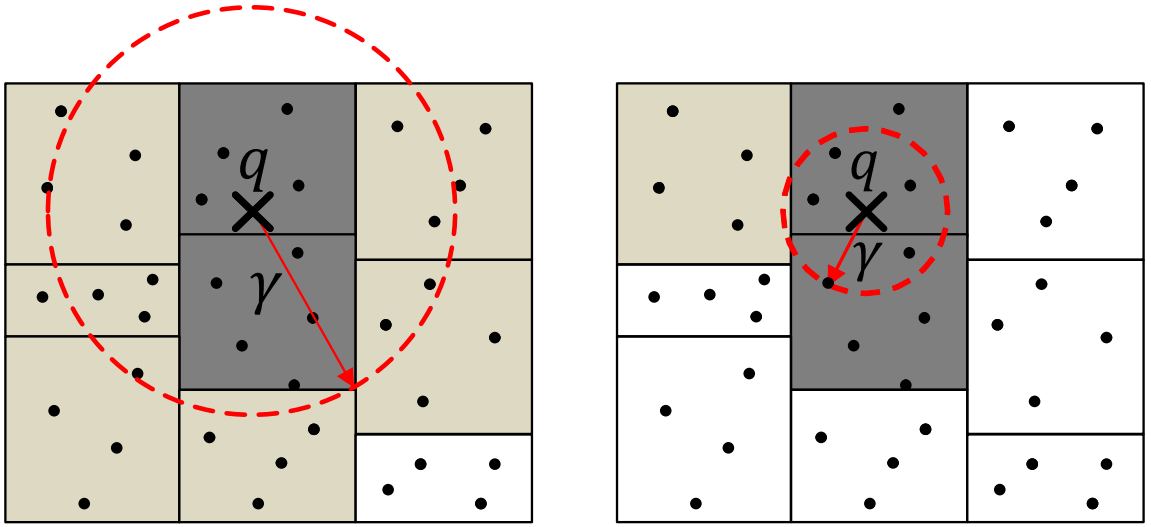
Range Queries

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



kNN Queries

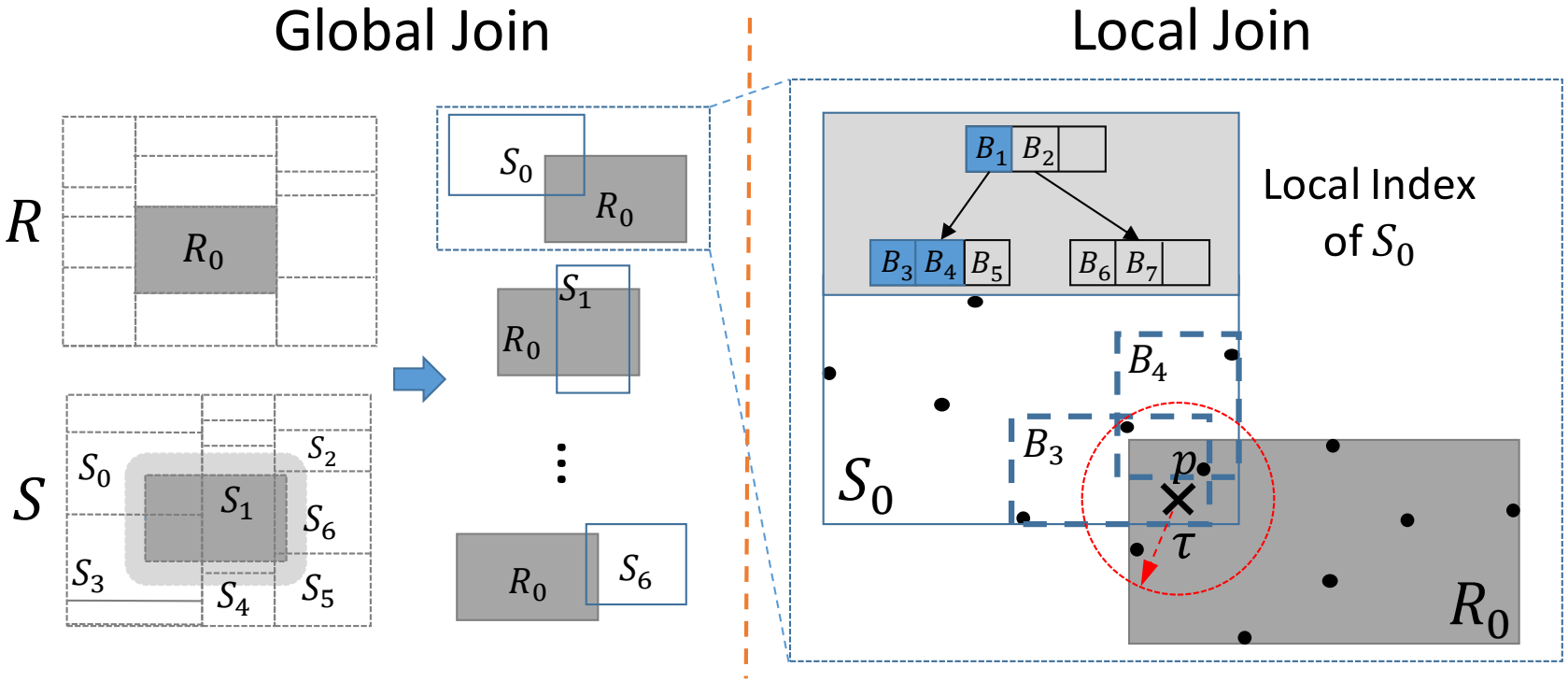
- $kNN(q, R)$
- Good performance thanks to:
  - Local indexes.
  - Pruning bound that is sufficient to cover global kNN results.



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

Distance Join

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



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

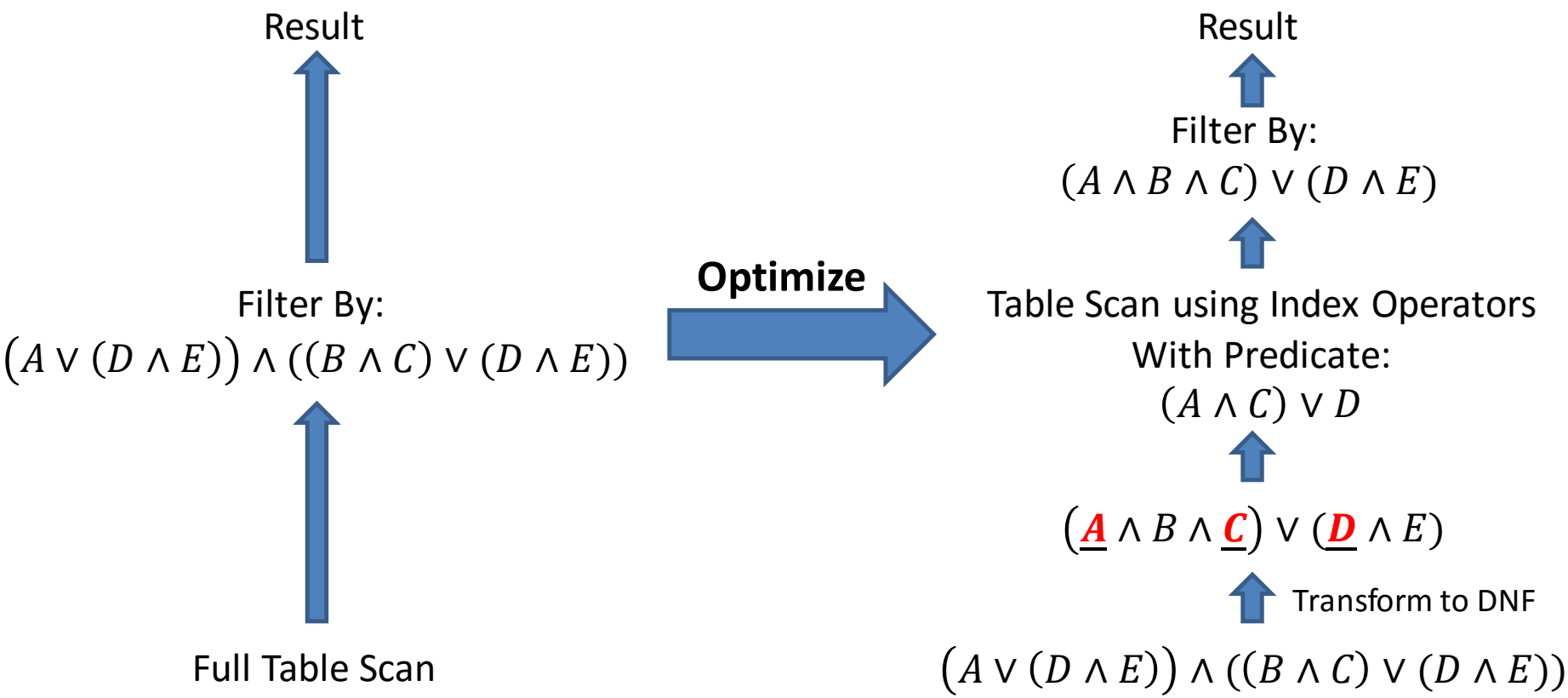
- $R \bowtie_{kNN} S$
- Explores several methods:
  - BKJSpark-N: Block nested loop *kNN* join in Spark.
  - BKJSpark-R: Block R-tree *kNN* join in Spark.
  - VKJSpark: Voronoi *kNN* join in Spark.
  - ZKJSpark: z-value *kNN* join in Spark.
  - RKJSpark: R-tree *kNN* join in Spark.

Agenda

Query Optimizer

- Index and geometry-awareness optimizations.
- Index scan optimization: for better index utilization.
- Selectivity estimation + Cost-based Optimization.
  - Selectivity estimation over local indexes
  - Choose a proper plan: scan or use index.
- Spatial predicates merging

Query Optimizer



Query Optimizer

- Partition size auto-tuning.
  - Data locality and load balancing through a good partitioner (STR Partitioner).
  - Memory fitness using record-size estimator.
- Broadcast join optimization: small table joins large table.
- Logical partitioning optimization for kNN joins.
  - Provides tighter pruning bounds.

Agenda

1. Background
2. Simba Architecture Overview
   - Programming Interface
   - Indexing
   - Spatial Operations
   - Optimization
3. Experiments
   - Comparison with Existing Systems
   - Comparison against Spark SQL
   - Distance and kNN Join Methods
4. Conclusions

Setup

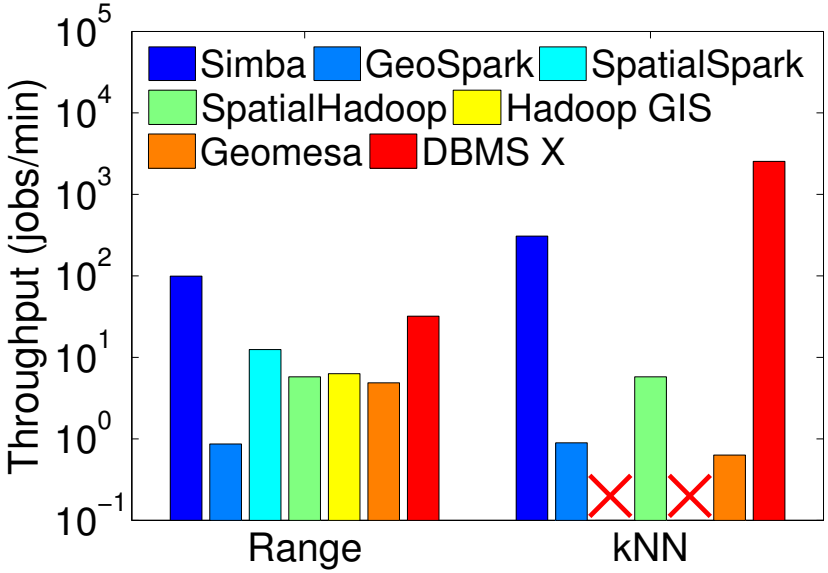
- 10 nodes cluster
- Processors: 6-core Intel Xeon E5 (1.6 to 2.0 GHz)
- RAM: 20 to 56 GB.
- Ubuntu 14.04 LTS, Hadoop 2.4.1, Spark 1.3.0

Datasets

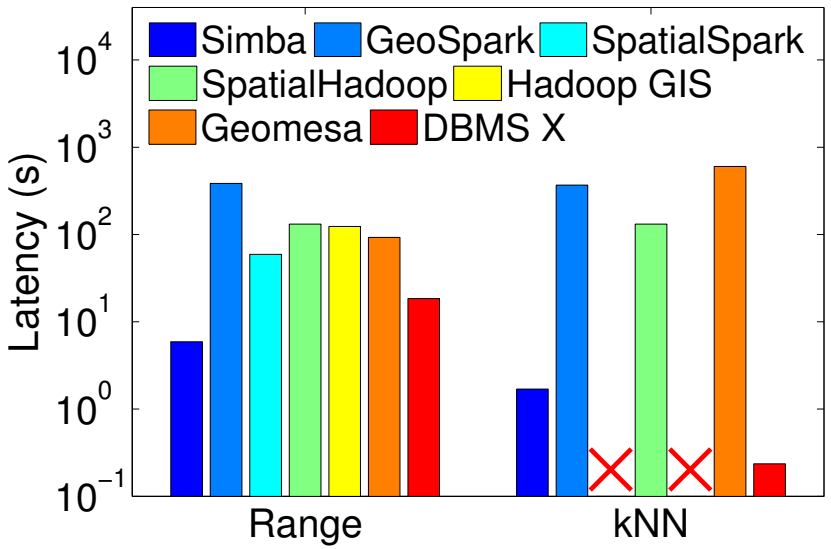
- OSM (OpenStreetMap)
  - 2.2 Billion records, 132GB.
  - Five fields: ID, a two-dimensional coordinate and two text information.
- GDEL (Global Data on Events, Language and Tone)
  - 75 Million records
  - Seven attributes: timestamp, three two-dimensional coordinates (start, end and action of the event).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

Agenda

Range and kNN Operations (OSM)

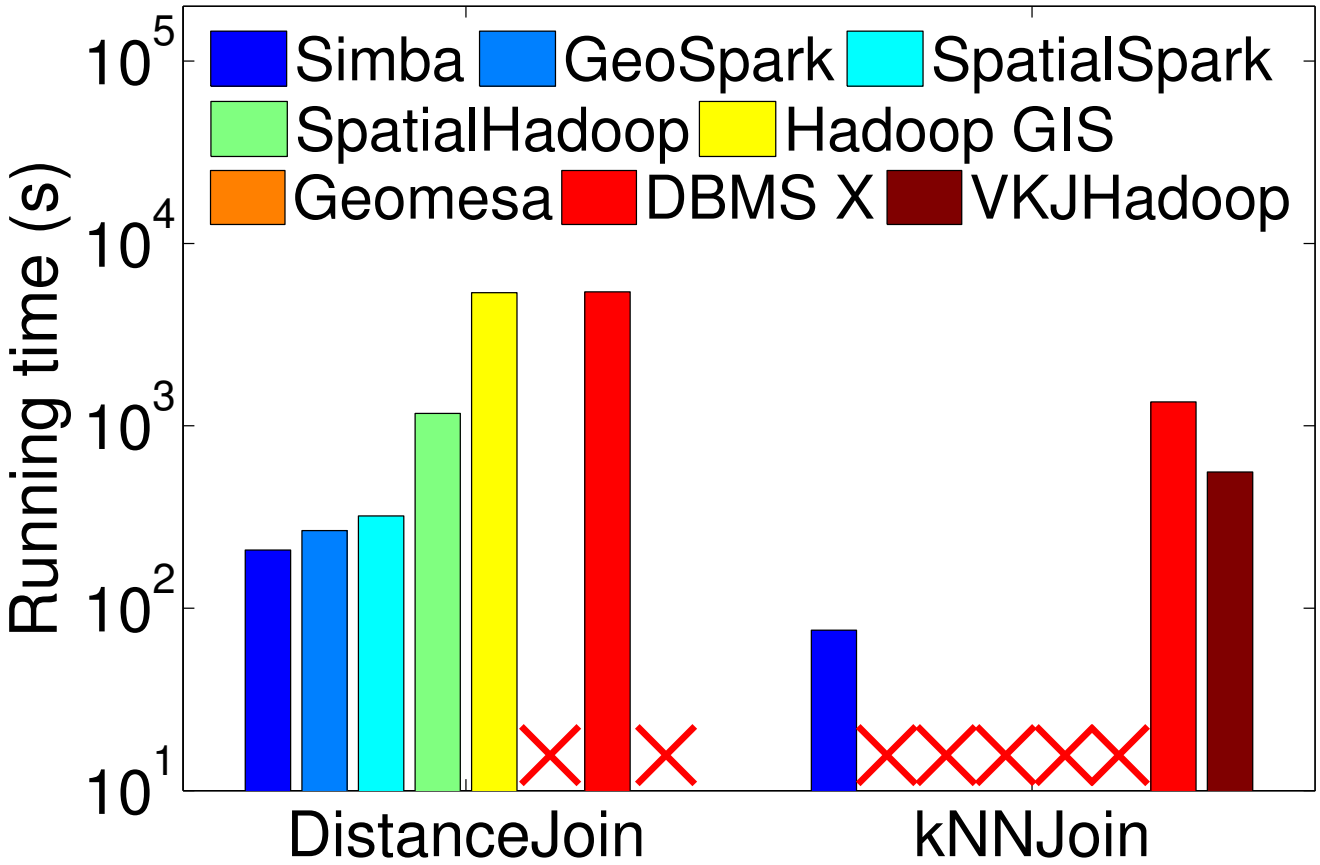


(a) Throughput

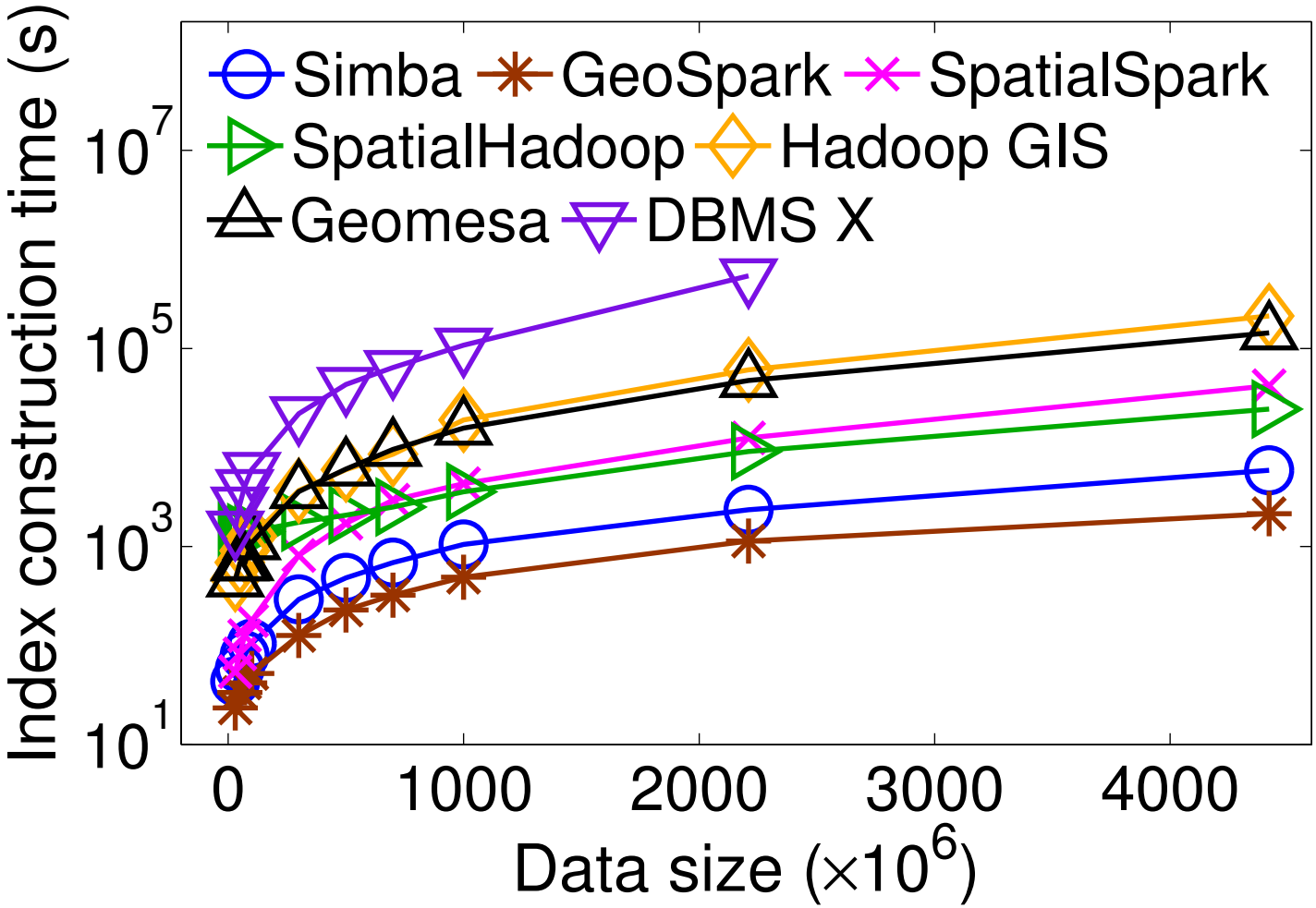


(b) Latency

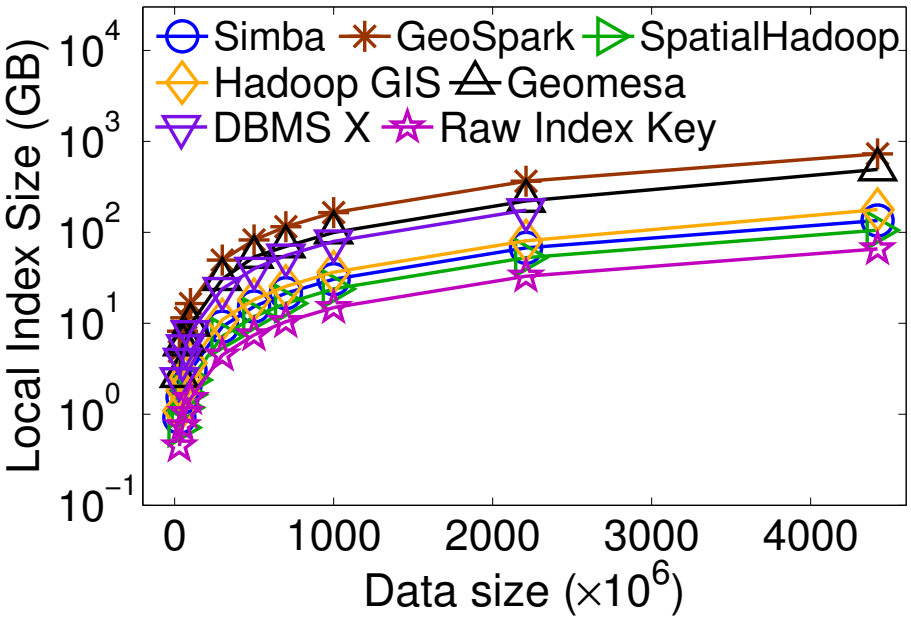
Join Operations (OSM)

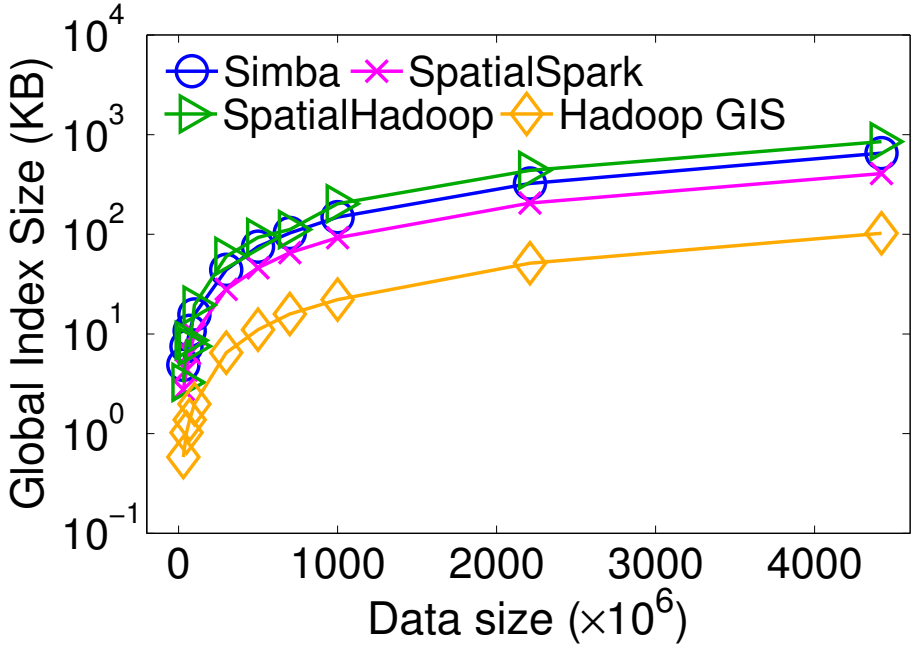


Index Construction Time (OSM)



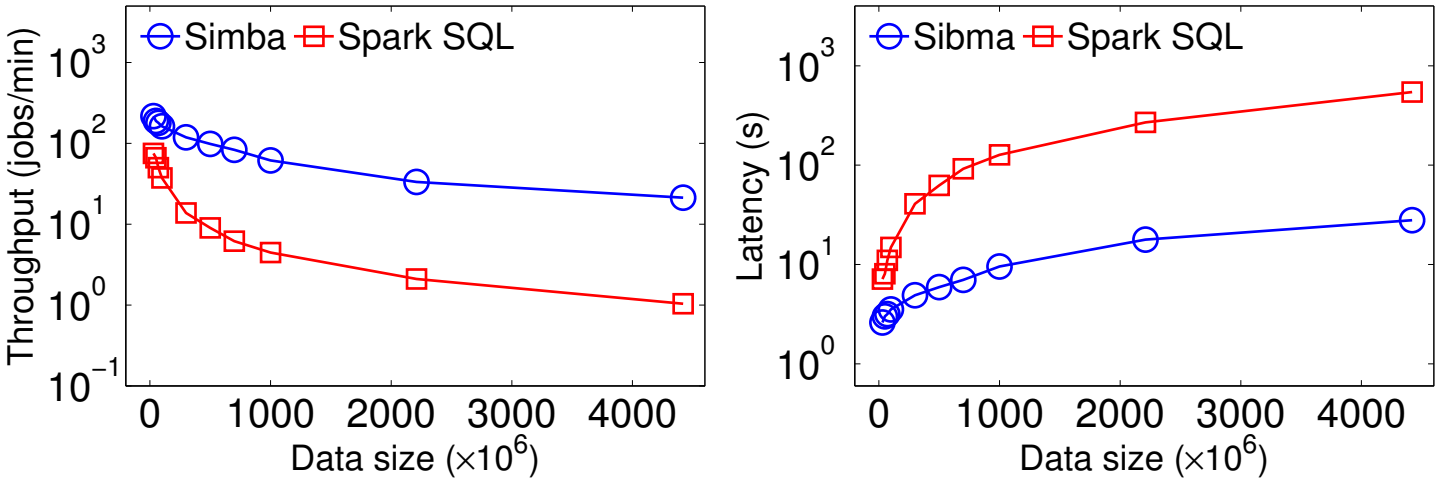
Index Storage Overhead (OSM)



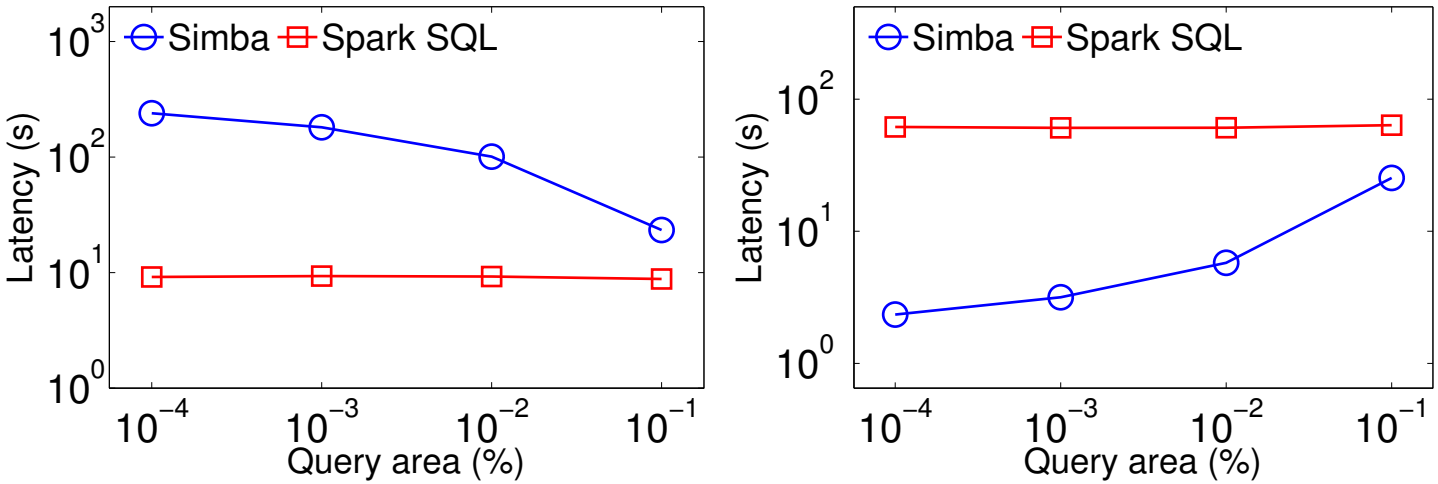


Agenda

Range Query Performance (OSM)

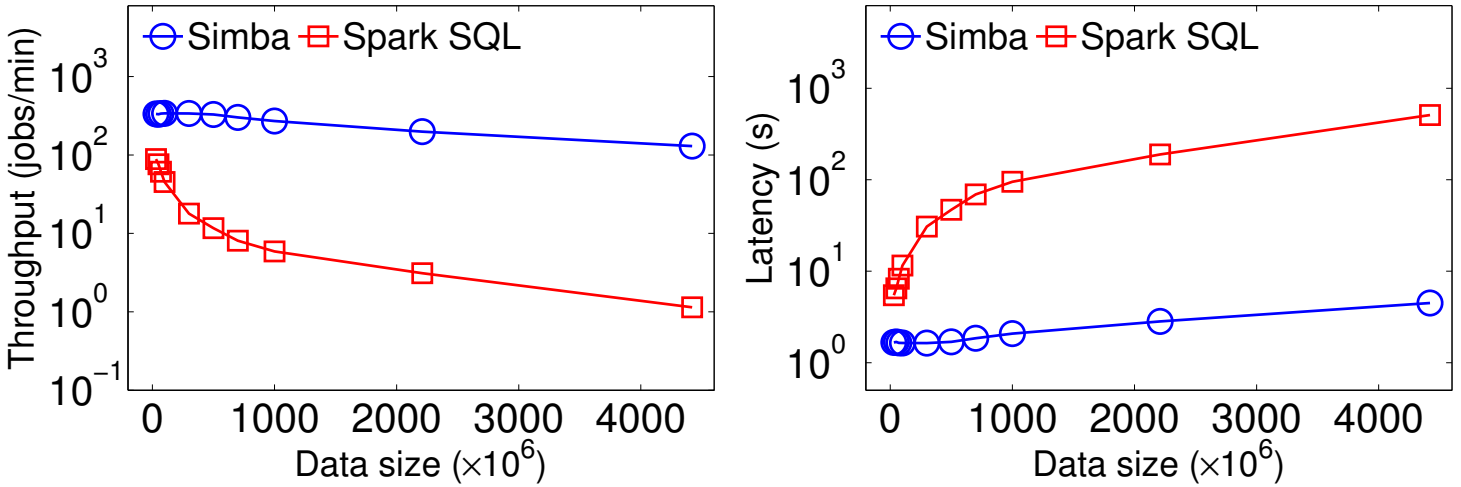


(a) Effect of data size.

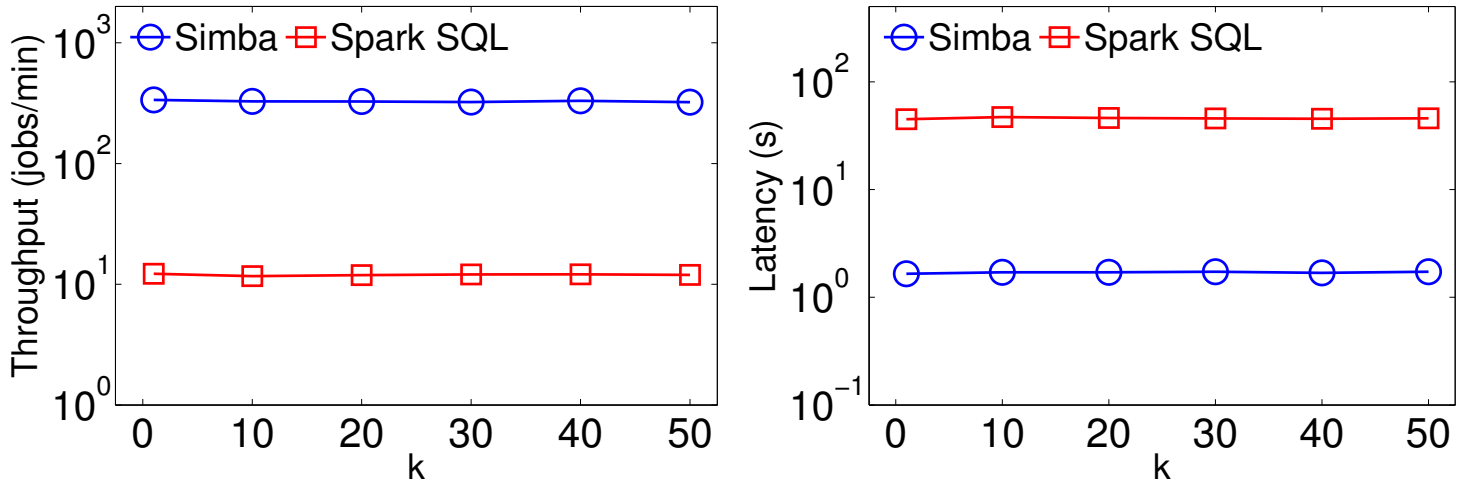


(b) Effect of query area size (percentage of total area).

kNN Query Performance (OSM)

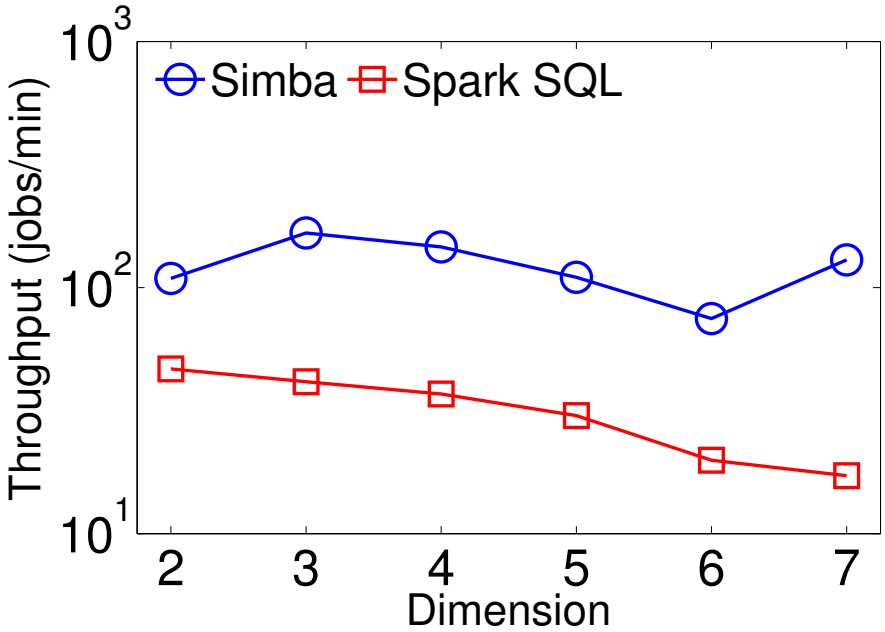


(a) Effect of data size.

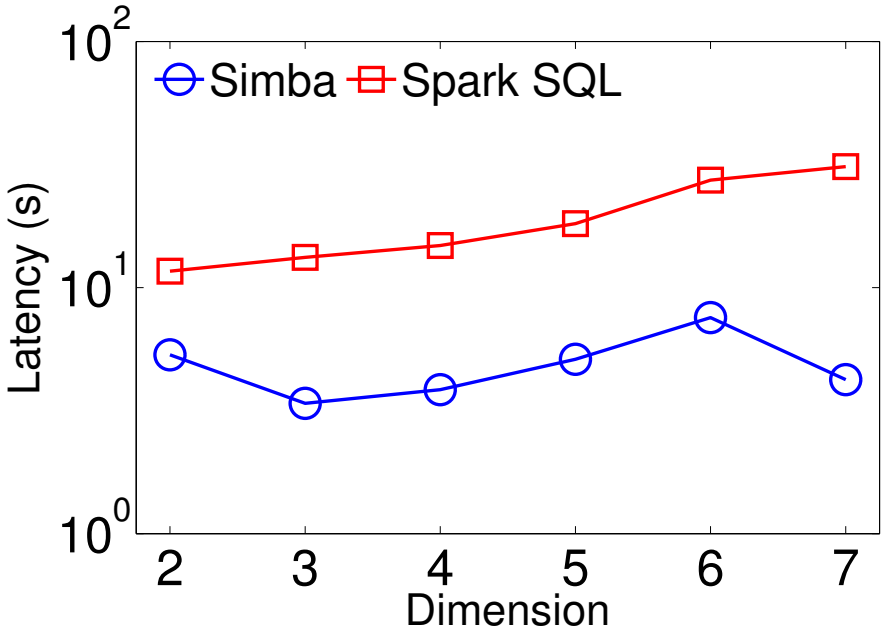


(b) Effect of k.

Range Query Performance (GDELT)

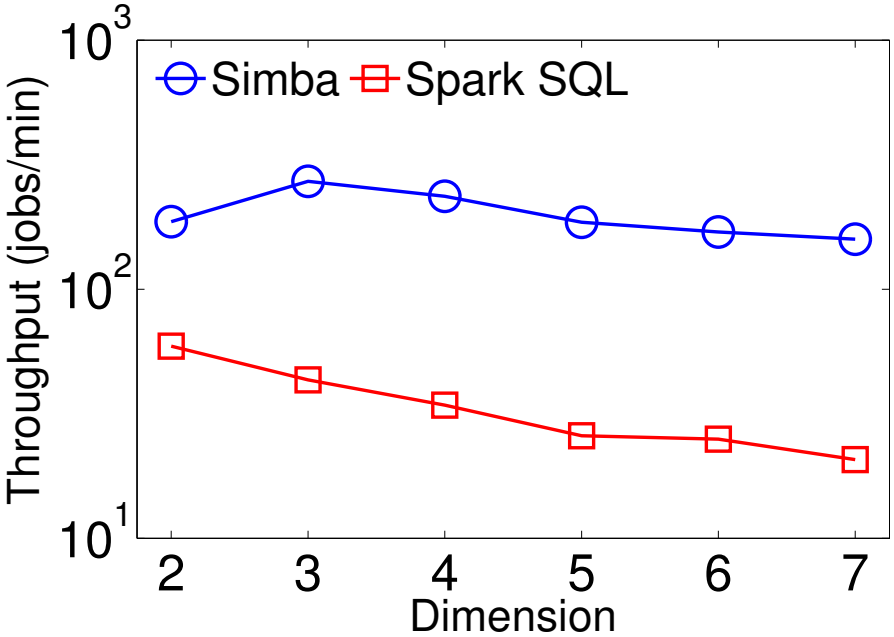


(a) Throughput

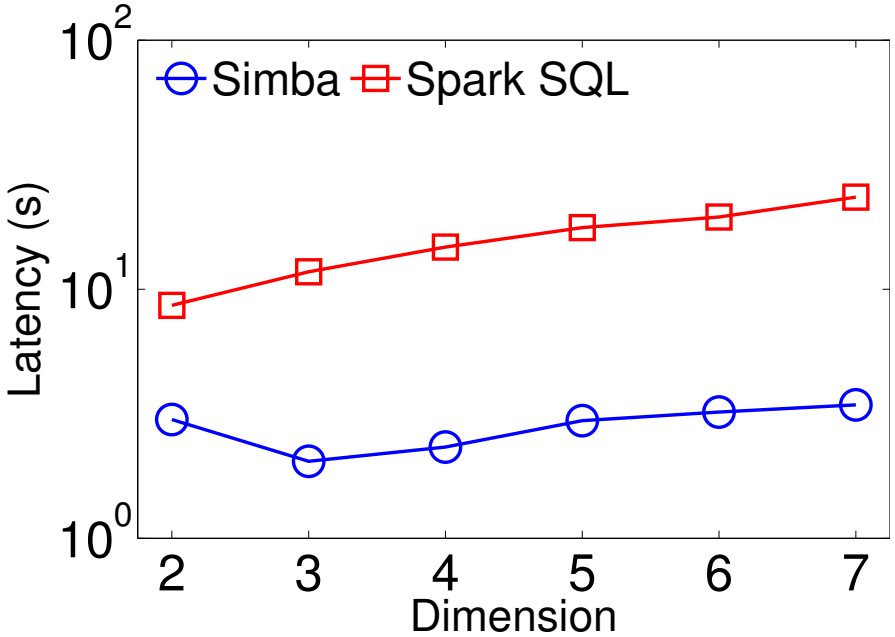


(b) Latency

kNN Query Performance (GDELT)



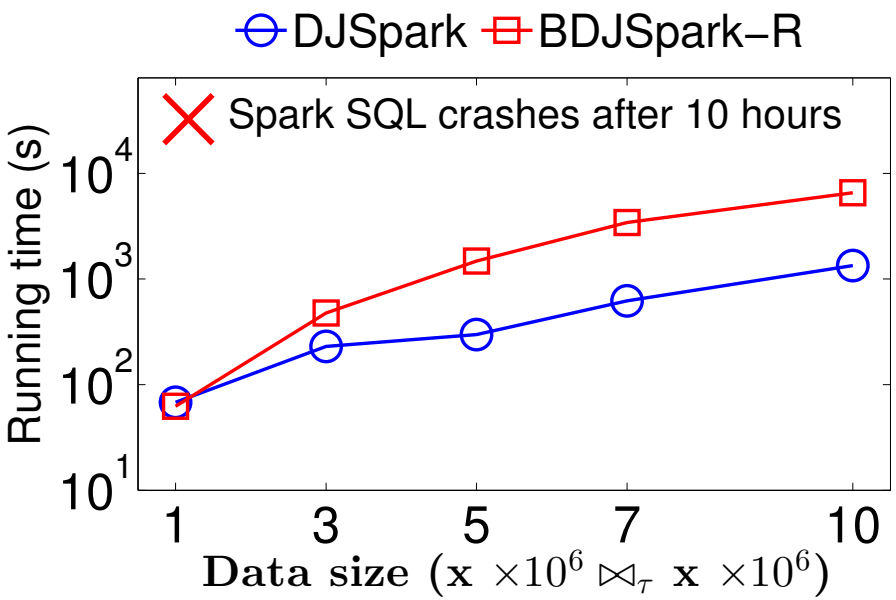
(a) Throughput



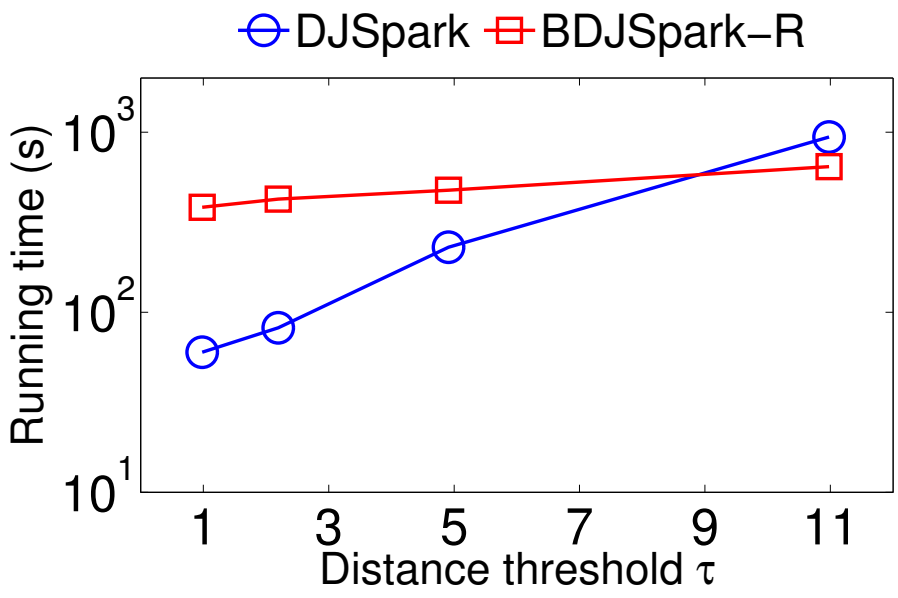
(b) Latency

Agenda

Distance Join Performance (OSM)

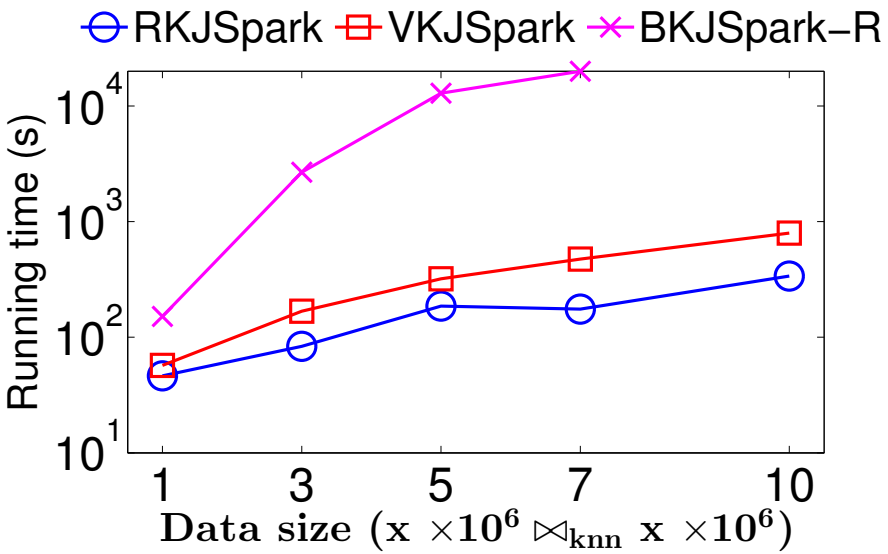


(a) Effect of data size.

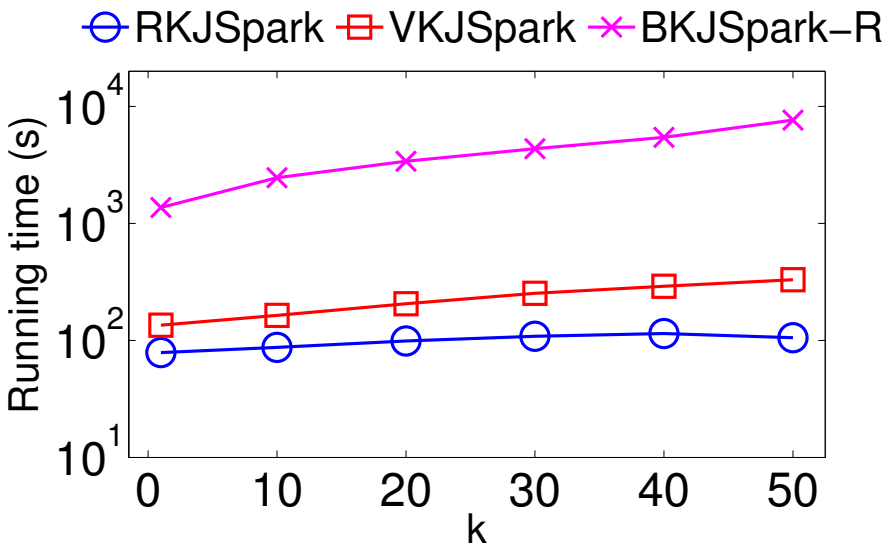


(b) Effect of τ.

kNN Join Performance (OSM)

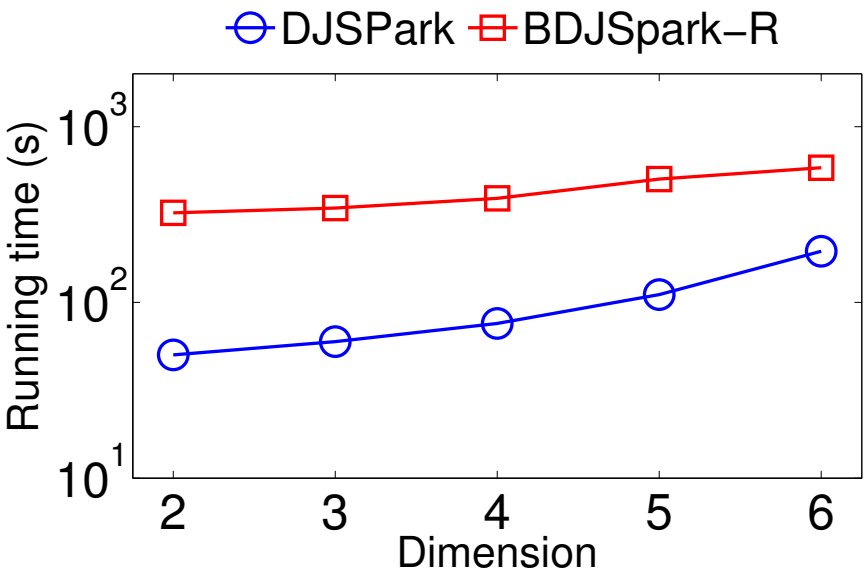


(a) Effect of data size.

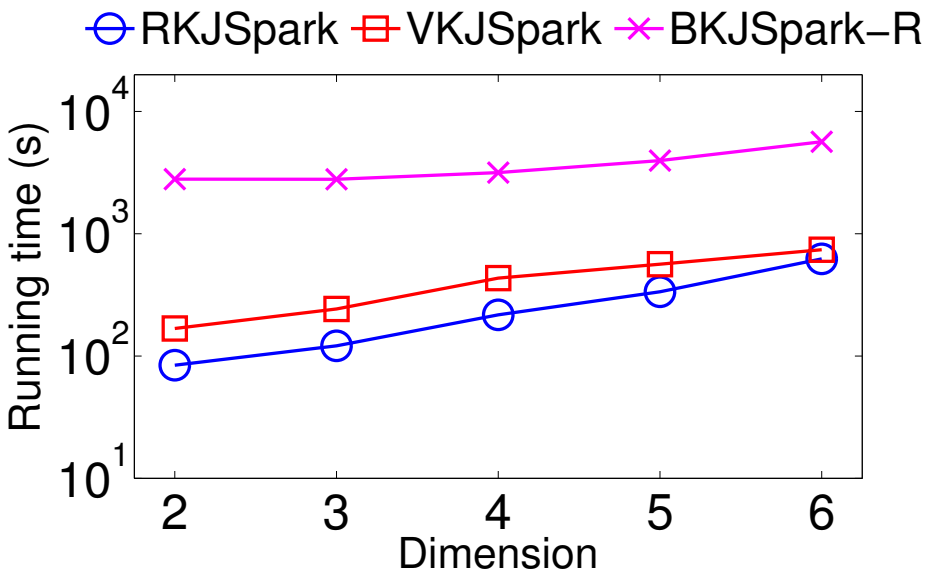


(b) Effect of k.

Join Operations Performance (RC)



(a) Distance join



(b) kNN join

Agenda

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

- Explore complex spatio-temporal patterns.
- Native support to geometry objects and operations.
- Spatial joins over predicates.

Future ideas

- Explore complex spatio-temporal patterns.
- Native support to geometry objects and operations.
- Spatial joins over predicates.

Future ideas

- Explore complex spatio-temporal patterns.
- Native support to geometry objects and operations.
- Spatial joins over predicates.

Future ideas

- Explore complex spatio-temporal patterns.
- Native support to geometry objects and operations.
- Spatial joins over predicates.

Thank you!!!

Do you have any question?