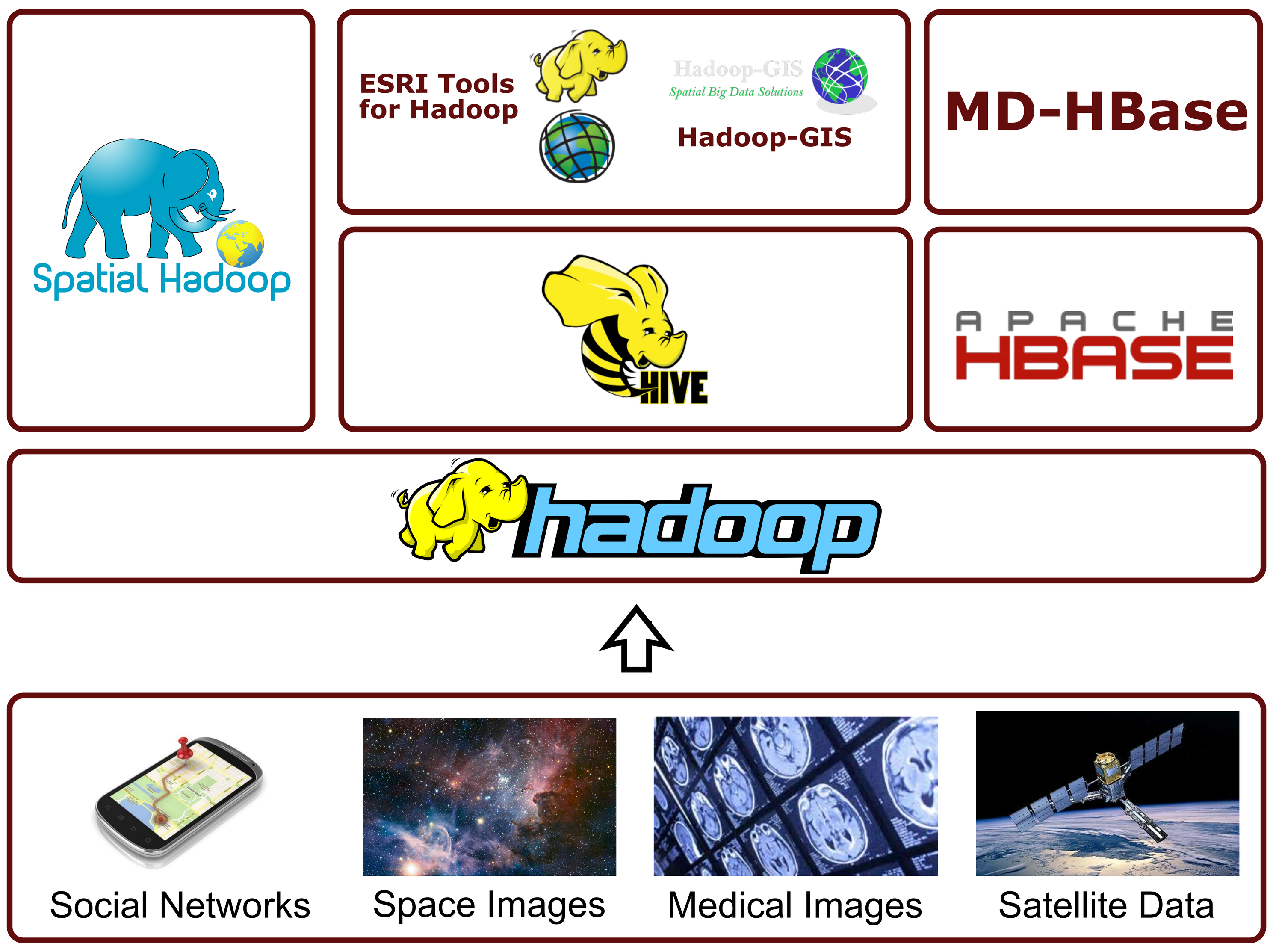
Sphinx

Distributed Execution of Interactive SQL Queries on Big Spatial Data

Ahmed Eldawy **Mohamed F. Mokbel**
Computer Science and Engineering
University of Minnesota

Mostafa Elganainy **Ammar Bakker** **Ahmed Abdelmotaleb**
KACST GIS Technology Innovation Center
Umm Al-Qura University, Saudi Arabia

Existing Big Spatial Data Systems



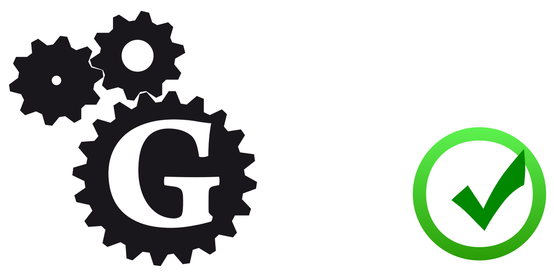
Limitaitons of Existing Systems

1. Lack of standard SQL query interface
2. Inherent limitations of they underlying systems (e.g., Hadoop)

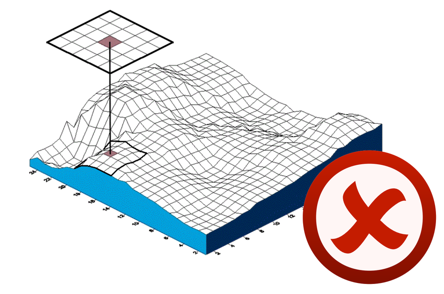
Cloudera Impala

SQL compliance

Query optimization

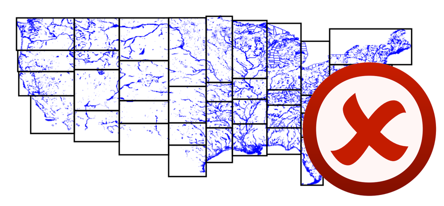


Runtime code generation



No spatial datatypes

Only supports relational datatypes such as numbers, Booleans and strings



No spatial indexes

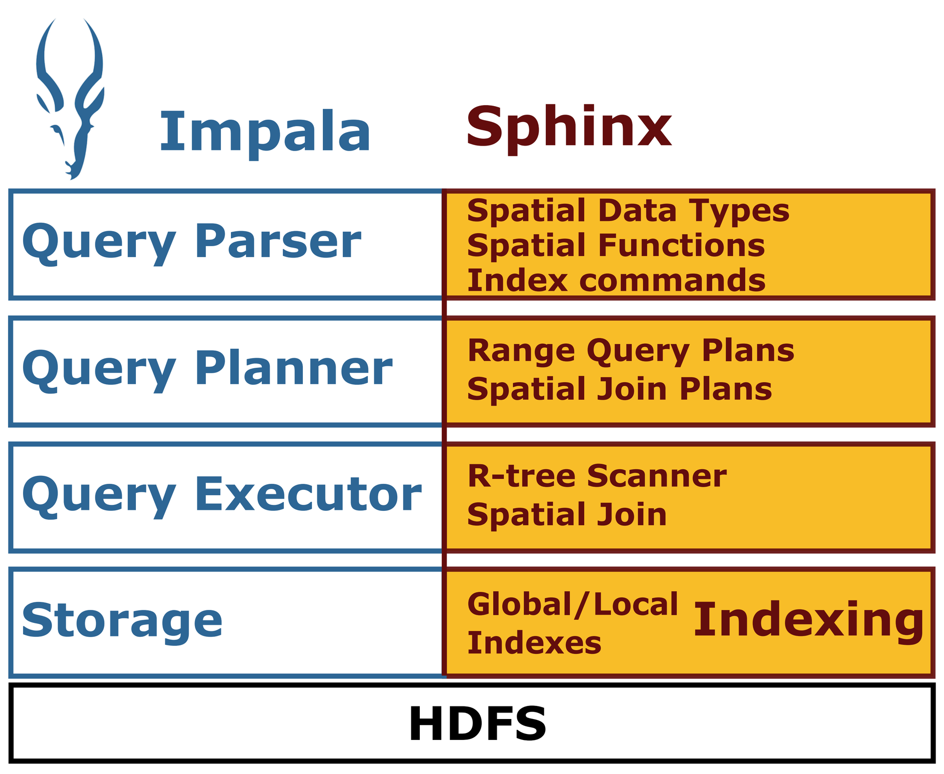
Spatial data is naturally skewed
No natural sorting order
Extended objects, like polygons, might overlap multiple partitions

No spatial operations

Impala only provides native query plans for simple selection or equi-joins, but lacks spatial operations such as range query or spatial join

Objective: Extend the core of Impala to support spatial data types, indexing and query processing efficiently

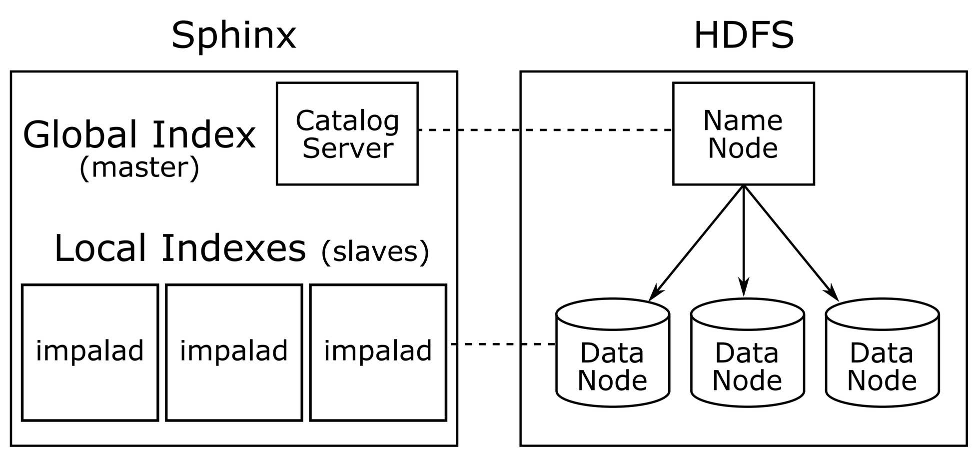
Sphinx Architecture



Query Parser

1. New GEOMETRY primitive data type
2. New spatial operations and spatial predicates
3. New CREATE INDEX command to construct R-tree and Quad tree
4. Extend CREATE EXTERNAL TABLE command to import SpatialHadoop indexes

Spatial Indexing



Global Index

Stays in the catalog server and stores how a table is partitioned into HDFS blocks

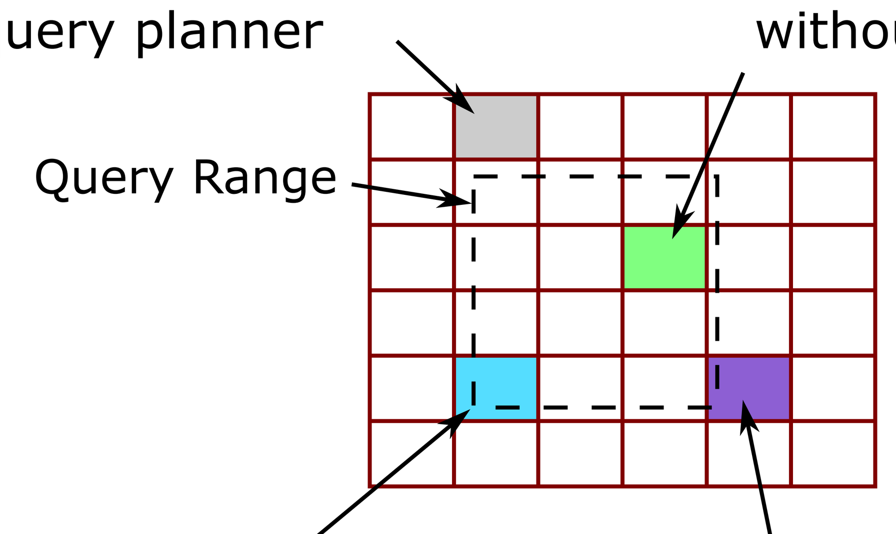
Local Indexes

Stored in slave nodes, as one local index per HDFS block. Determines how records are organized inside each HDFS block

Range Query

Case 0
A partition is completely outside the query range
Early pruned by the query planner

Case 1
A partition is completely contained in the query range
All records are returned without further processing

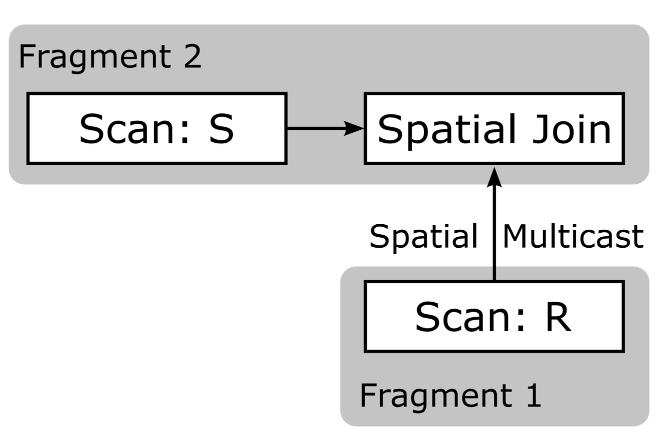


Case 2
Most of the partition overlaps the query range
Skip the local index and scan all records in the partition

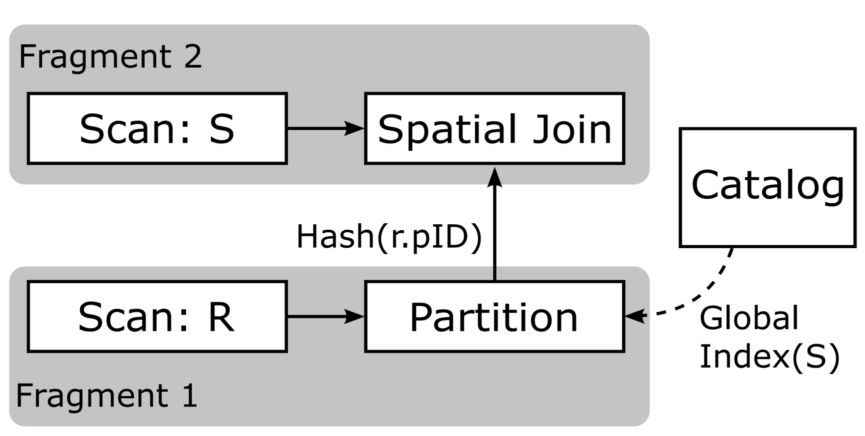
Case 3
A small portion of the partition overlaps the query range
Use the local index to speed up the range query processing

The optimized code is produced using runtime code generation

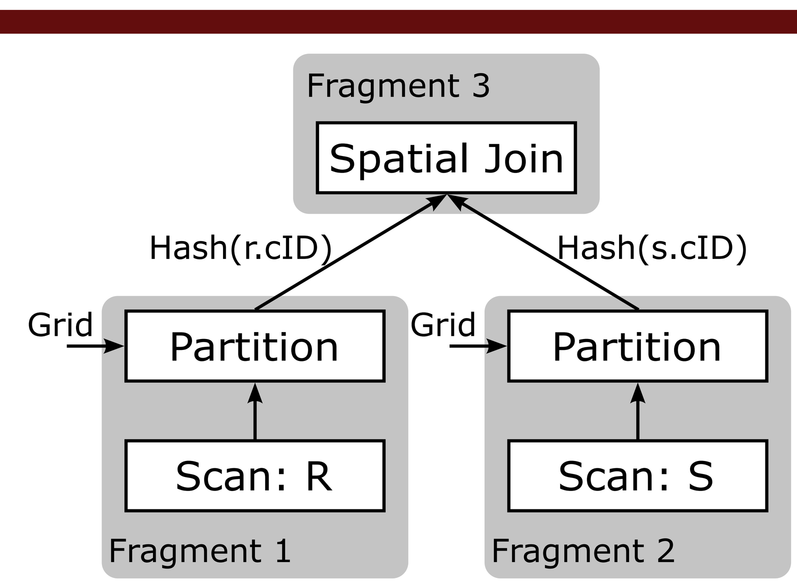
Spatial Join Plans



Two indexes (Overlap join)
The query planner finds overlapping partitions
The query executor joins every pair of partitions

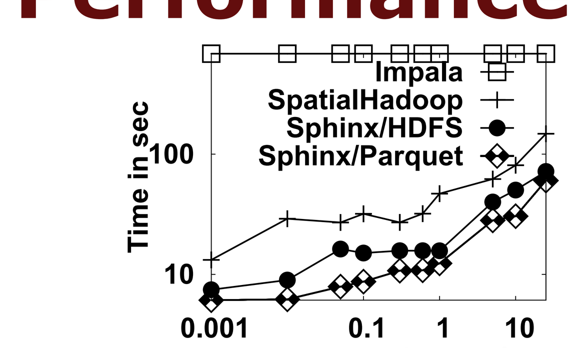


One indexes (Partition join)
Partition the non-indexed table to match the indexed one
Join each pair of matching partitions

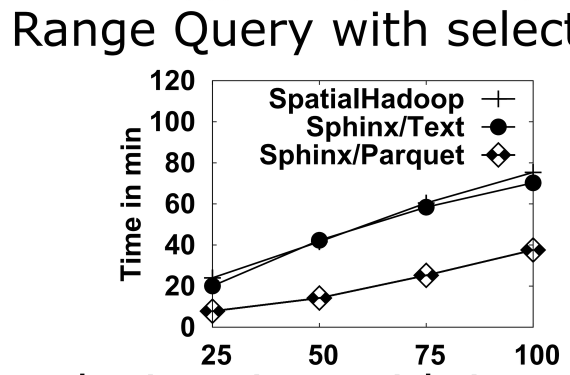


No indexes (Co-partition join)
Co-partition the two files using a common grid
Match the contents of each grid cell

Performance

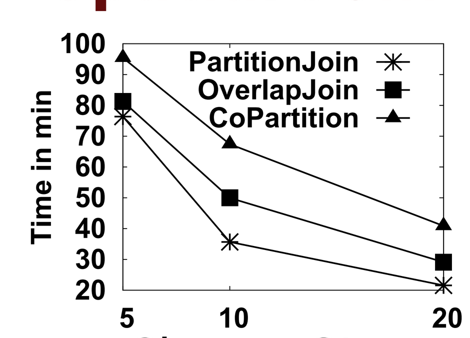


Range Query with selectivity

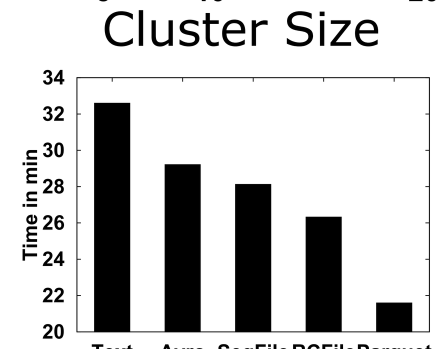


Indexing time with input size

Spatial Join



Cluster Size



Storage formats

This work is supported in part by KACST GIS Technology Innovation Center at Umm Al-Qura University, Saudi Arabia