RIOS: Runtime Integrated Query Optimizer for Spark

Youfu Li(UCLA), Mingda Li(UCLA), Ling Ding(UCLA) and Matteo Interlandi(Microsoft)
Cloud Computing Programs

Open Source Data-Intensive Scalable Computing (DISC) Platforms: Hadoop MapReduce and Spark

- functional API
- map and reduce User-Defined Functions
- RDD transformations (filter, flatMap, zipPartitions, etc.)

Several years later, introduction of high-level SQL-like declarative query languages (and systems)

- Conciseness
- Pick a physical execution plan from a number of alternatives
Query Optimization

Two steps process
- **Logical** optimizations (e.g., filter pushdown)
- **Physical** optimizations (e.g., join orders and implementation)

Physical optimizer in RDMBS:
- **Cost-based**
- **Data statistics** (e.g., predicate selectivities, cost of data access, etc.)

The role of the cost-based optimizer is to
1. enumerate some set of equivalent plans
2. estimate the cost of each
3. select a sufficiently good plan
Query Optimization: Why Important?

Spark
AsterixDB
Hive
Pig

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

unable to finish in 5+ hours

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

unable to finish in 5+ hours

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

unable to finish in 5+ hours

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

unable to finish in 5+ hours

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

unable to finish in 5+ hours

Time (s)

Scale Factor = 10

Spark
AsterixDB
Hive
Pig

unable to finish in 5+ hours
Query Optimization: Why Important?

Bad plans over Big Data can be disastrous!
Challenges for Cost-based Optimizer in DISC

Lack of *upfront statistics*:
- data sits in HDFS and unstructured

Even if input statistics are available:
- *Correlations* between predicates
- Exponential error propagation in joins
- Arbitrary *UDFs*
Cost-based Optimizer in DISC: State of the Art

Pre-existing statistics

Bad statistics

No upfront statistics
Cost-based Optimizer in DISC: State of the Art

**Pre-existing statistics**
- Spark CBO[1]

**Bad statistics**

**No upfront statistics**

- Collect and store statistics
- No runtime plan revision
Cost-based Optimizer in DISC: State of the Art

**Pre-existing** statistics
- Spark CBO[1]

**Bad** statistics
- Adaptive Query planning[2]

**No upfront** statistics

- Collect and store statistics
- No runtime plan revision

Assumption is that some initial statistics exist
Cost-based Optimizer in DISC: State of the Art

**Pre-existing statistics**
- Spark CBO[1]

**Bad statistics**
- Adaptive Query planning[2]

**No upfront statistics**
- Pilot runs (samples)[3]

- Collect and store statistics
- No runtime plan revision

Assumption is that some initial statistics exist

- Samples are expensive
- Only foreign-key joins
- No runtime plan revision
Traditional Query Planning VS RIOS

Logical Plan

Query Optimizer

Physical Plan

Execution

Statistics
Traditional Query Planning VS RIOS

Query Optimizer -> Logical Plan
                    |                          |
                    v                          v
Physical Plan       Execution               Planning
                    |                          |
                    v                          v
Logical Plan         Statistics          Physical plan

RIOS
      Execution
      Runtime Stats
Runtime Integrated Optimizer for Spark

Key idea: Execute-Gather-Aggregate-Plan strategy (EGAP)
- Query plans are lazily executed
- Statistics are gathered at runtime
- **Aggregate** statistics after gathering
- Joins are greedily planned for execution
- Plan can be **dynamically changed** if a **bad decision** was made

Neither upfront statistics nor pilot runs are required
- Raw dataset size is required for initial guess

Support for not foreign-key joins
Runtime Optimizer: an Example

A \not< B \not< C

Assumption: A < C
Runtime Optimizer: Execute Step

Assumption: A < C
Runtime Optimizer: Gather step

Assumption: A < C
Runtime Optimizer: Aggregate step

Assumption: A < C
Runtime Optimizer: Plan step

Assumption: A < C
Runtime Optimizer: Execute step

Assumption: A < C
Runtime Optimizer: Gather step

Assumption: A < C
Runtime Optimizer: Plan step

Assumption: A < C
Runtime Optimizer: Execute step

Assumption: $A < C$
Runtime Optimizer: Wrong Guess

\[ \sigma(A) \times B \times \sigma(C) \]

Assumption: \( A < C \)

\[ \sigma(A) > \sigma(C) \]
Runtime Optimizer: Wrong Guess

\[ \sigma(A) \text{ } \Box \text{ } B \text{ } \Box \text{ } \sigma(C) \]

Assumption: \( A < C \)

\[ \sigma(A) > \sigma(C) \]
Runtime Optimizer: Wrong Guess

Assumption: $A < C$

$\sigma(A) \land B \land \sigma(C)$

$\sigma(A) > \sigma(C)$

Repartition Step
Runtime Optimizer: Wrong Guess

Assumption: $A < C$

$\sigma(A) \otimes B \otimes \sigma(C)$

$\sigma(A) > \sigma(C)$
Runtime Optimizer: Wrong Guess

\[ \sigma(A) \Join B \Join \sigma(C) \]  
Assumption: \( A < C \)  
\[ \sigma(A) > \sigma(C) \]
Runtime Optimizer: Wrong Guess

Assumption: $A < C$

$\sigma(A) \geq \sigma(C)$

$\sigma(A) > \sigma(C)$
Runtime Optimizer: Wrong Guess

Assumption: $A < C$

$\sigma(A) > \sigma(C)$
Runtime Integrated Optimizer for Spark

Spark **batch execution** model allows late binding of joins

Set of Statistics:
  ◦ Join estimations (based on sampling or sketches)
  ◦ Number of records
  ◦ Average size of each record

Statistics are aggregated using a Spark job or accumulators

Join **implementations** are picked based on thresholds
Challenges and Optimizations

Execute - Block and revise execution plans without wasting computation

Aggregate - Efficient accumulation of statistics

Plan - Try to schedule as many broadcast joins as possible
Experiments

Q1: What are the performance of RIOS compared to regular Spark, pilot runs and Spark-CBO?

Q2: How expensive are wrong guesses?
Minibenchmark with 3 Fact Tables

Time (s)

spark good-order
RIOS R2R
RIOS W2R
pilot-run
spark wrong-order
unable to finish in 5+ hours
unable to finish in 5+ hours

Scale Factor

1
10
100
1000

11
9
11
174
69
25
24
25
1312
1242
92
86
93
9095
8951
1482
1353
1521

Minibenchmark with 3 Fact Tables

Q1: RIOS is always faster than Spark and pilot run.
Minibenchmark with 3 Fact Tables

Q2: Not much, around 15% in the worst case
TPCDS and TPCH Queries

![Bar chart showing query times for different scale factors and query numbers, with query times in seconds for spark good-order, CBO-plan, RIOS pilot-run, and spark bad-order.]

- Query 17
- Query 50
- Query 28
- Query 9

Time (s)

Scale Factor

Unable to finish in 5+ hours
Q1: RIOS is always the faster approach
Conclusions

RIOS: cost-based query optimizer for Spark

Statistics are **gathered at runtime** (no need for initial statistics or pilot runs)

**Late bind** of joins

Up to **2x faster** than the best plan generated by pilot run, and **> 100X** than previous approaches for fact table joins.
Thank you
Experiment Configuration

- **Datasets:**
  - TPCDS
  - TPCH

- **Configuration:**
  - 16 machines, 4 cores (2 hyper threads per core) machines, 32GB of RAM, 1TB disk
  - Spark 2.2.1
  - Scale factor from 1 to 1000 (~1TB)
Reference

Thank you