Cost-based Optimizer Framework for Spark SQL

Ron Hu, Zhenhua Wang

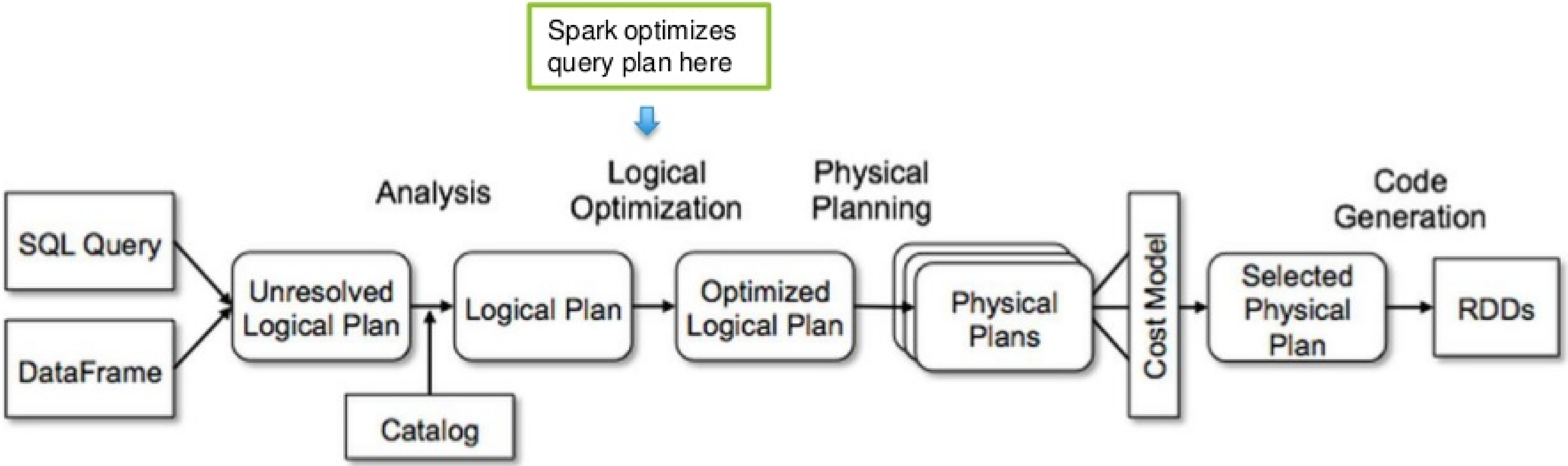
Huawei Technologies

SPARK SUMMIT EAST 2017

Presentation Overview

- Catalyst Architecture
- Rule-based Optimizations
- Reliable Statistics Collected
- Cardinality Estimation
- Cost-based Optimizations
- Explain Enhancement
- Performance Results
- Future Work
- Q & A

Catalyst Architecture



Reference:[Deep Dive into Spark SQL's Catalyst Optimizer](), a databricks engineering blog

SPARK SUMMIT EAST 2017

Rule-based Optimizer in Spark SQL

- Most of Spark SQL optimizer's rules are heuristics rules.
  - PushDownPredicate, ColumnPruning, ConstantFolding,….
- Does NOT consider the cost of each operator
- Does NOT consider filter factor when estimating join relation size
- Join order is decided by its position in the SQL queries
- Join algorithm selection is decided by some very simple system assumptions

Birth of Spark SQL CBO

- Prototype
  - In 2015, Ron Hu, Fang Cao, etc. of Huawei's research department prototyped the CBO concept on Spark 1.2.
  - After a successful prototype, we shared technology with Zhenhua Wang, Fei Wang, etc of Huawei's product development team.
- We delivered a talk at Spark Summit 2016:
  - "Enhancing Spark SQL Optimizer with Reliable Statistics".
- The talk was well received by the community.
  - https://issues.apache.org/jira/browse/SPARK-16026

Phase Delivery

- In the first CBO release, we plan to contribute Huawei's existing CBO code to community.
  - It is a good and working CBO framework to start with.
- Focus on
  - Statistics collection,
  - Cardinality estimation,
  - Build side selection, broadcast vs. shuffled join, join reordering, etc.
- Will use heuristics formula for cost function.

Statistics Collected

- Collect Table Statistics information
- Collect Column Statistics information
- Goal:
  - Calculate the cost for each operator in terms of number of output rows, size of output, etc.
  - Based on the cost calculation, adjust the query execution plan

SPARK SUMMIT EAST 2017

Table Statistics Collected

- Command to collect statistics of a table.
  - Ex: `ANALYZE TABLE table-name COMPUTE STATISTICS`
- It collects table level statistics and saves into metastore.
  - Number of rows
  - Table size in bytes

SPARK SUMMIT EAST 2017

Column Statistics Collected

- Command to collect column level statistics of individual columns.
  - Ex: `ANALYZE TABLE table-name COMPUTE STATISTICS FOR COLUMNS column-name1, column-name2, ….`
- It collects column level statistics and saves into meta-store.

- ☐ Numeric/Date/Timestamp type
  - ✓ Distinct count
  - ✓ Max
  - ✓ Min
  - ✓ Null count
  - ✓ Average length (fixed length)
  - ✓ Max length (fixed length)

- ☐ String/Binary type
  - ✓ Distinct count
  - ✓ Null count
  - ✓ Average length
  - ✓ Max length

Filter Cardinality Estimation

- Between Logical expressions: AND, OR, NOT
- In each logical expression: =, <, <=, >, >=, in, etc
- Current support type in Expression
  - For <, <=, >, >=: Integer, Double, Date, Timestamp, etc
  - For =: String, Integer, Double, Date, Timestamps, etc.
- Example: A <= B
  - Based on A, B's min/max/distinct count/null count values, decide the relationships between A and B. After completing this expression, we set the new min/max/distinct count/null count
  - Assume all the data is evenly distributed if no histogram information.

SPARK SUMMIT EAST 2017

Filter Operator Example

- ***Column A (op) literal B***
  - (op) can be “=“, “<”, “<=”, “>”, “>=”, “like”
  - Like the styles as “l_orderkey = 3”, “l_shipdate <= “1995-03-21”
  - Column’s max/min/distinct count/null count should be updated
  - Example: **Column A < value B**

Filtering Factor = 100%
no need to change A’s statistics

Filtering Factor = 0%
need to change A’s statistics

Without histograms, suppose data is evenly distributed
Filtering Factor = (B.value – A.min) / (A.max – A.min)
A.min = no change
A.max = B.value
*A.ndv = A.ndv * Filtering Factor*

SPARK SUMMIT EAST 2017

Filter Operator Example

- ***Column A (op) Column B***
  - (op) can be "<", "<=", ">", ">="
  - We cannot suppose the data is evenly distributed, so the empirical filtering factor is set to **1/3**
  - Example: **Column A < Column B**

A filtering = 100%
B filtering = 100%

A filtering = 0%
B filtering = 0%

A filtering = 33.3%
B filtering = 33.3%

A filtering = 33.3%
B filtering = 33.3%

Join Cardinality Estimation

- *Inner-Join:* The number of rows of "A join B on A.k1 = B.k1" is estimated as: T(A ***IJ*** B) = T(A) * T(B) / max(V(A.k1), V(B.k1)),
  - where T(A) is the number of records in table A, V is the number of distinct values of that column.
  - The underlying assumption for this formula is: each value of the smaller domain is included in the larger domain.
- *Left-Outer Join*: T(A ***LOJ*** B) = max (T(A ***IJ*** B) , T(A))
- *Right-Outer Join*: T(A ***ROJ*** B) = max (T(A ***IJ*** B) , T(B))
- *Full-Outer Join*: T(A ***FOJ*** B) = T(A ***LOJ*** B) + T(A ***ROJ*** B) - T(A ***IJ*** B)

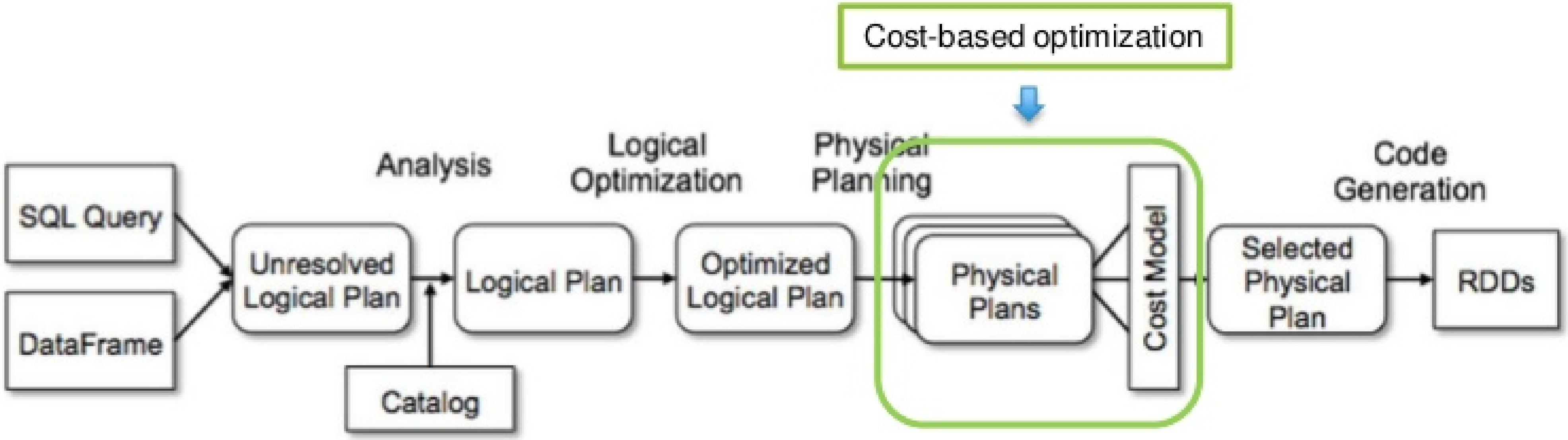
Other Operator Estimation

- Project: does not change row count
- Aggregate: consider uniqueness of group-by columns
- Limit
- Sample
- …

SPARK SUMMIT EAST 2017

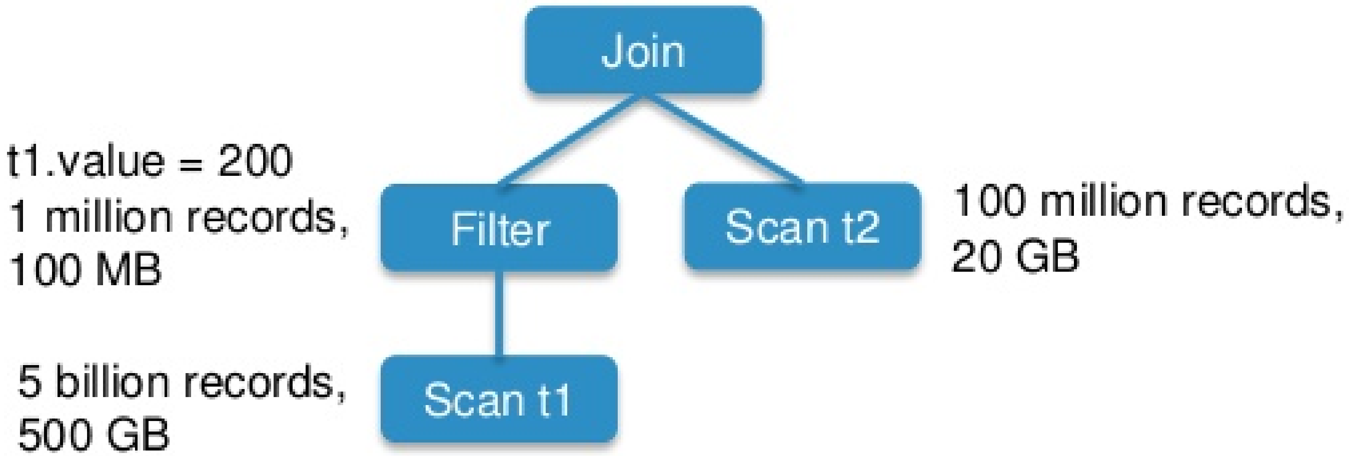
Cost-based Optimizations

- Choose the best physical plan based on cost.



Build Side Selection

- For two-way hash joins, we need to choose one operand as build side and the other as probe side.
- We calculate the cost of left and right sides in hash join.
  - Nominal Cost = <nominal-rows> × 0.7 + <nominal-size> × 0.3
- Choose lower-cost child as build side of hash join.
  - Before: build side was selected based on original table sizes. ➔ BuildRight
  - Now with CBO: build side is selected based on estimated cost of various operators before join. ➔ BuildLeft



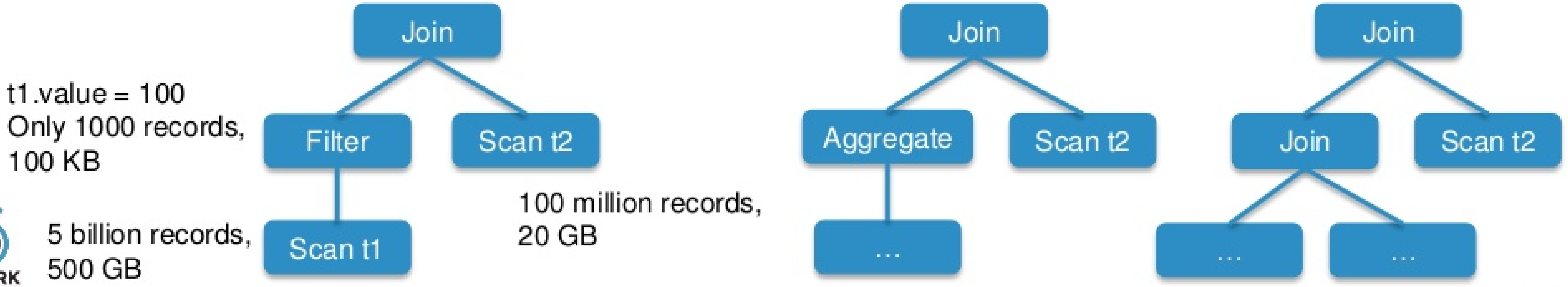
SPARK SUMMIT EAST 2017

Hash Join Implementation: Broadcast vs. Shuffle

- **Logical Plan**
  - Equi-join
    - Inner Join
    - LeftSemi/LeftAnti Join
    - LeftOuter/RightOuter Join
  - Theta-join

- **Physical Plan**
  - ShuffledHashJoinExec/ BroadcastHashJoinExec
  - CartesianProductExec/ BroadcastNestedLoopJoinExec

- Broadcast criterion: whether the join side's output size is small (default 10MB).



Multi-way Join Reorder

- Currently Spark SQL's Join order is not decided by the cost of multi-way join operations.
- We decide the join order based on the output rows and output size of the intermediate tables.
  - Use a combination of heuristics and dynamic programming.
  - Use statistics to derive if a join attribute is unique.
  - Can benefit star join queries (like TPC-DS).
  - Consider shuffle cost.
  - Still under development.

Explain Enhancement

- EXPLAIN STATS statement displays statistics for each operator in the optimized logical plan:
  - Size in bytes, row count, broadcast hint, etc.
- Example:

```
> EXPLAIN  STATS
> SELECT cc_call_center_sk, cc_call_center_id, cc_rec_start_date FROM call_center;
…
== Optimized Logical Plan ==
Project [cc_call_center_sk#5127, cc_call_center_id#5128, cc_rec_start_date#5129],
            Statistics(sizeInBytes=352.0 B, rowCount=8, isBroadcastable=false)
+- Relation[…fields] parquet, Statistics(sizeInBytes=15.8 KB, rowCount=8, isBroadcastable=false)
…
```

SPARK SUMMIT EAST 2017

Preliminary Performance Test

- Setup:
  - TPC-DS size at 2 TB (scale factor 2000)
  - 4 node cluster (40 cores, 380GB mem each)
  - Latest Spark development code

- Statistics collection
  - A total of 24 tables and 425 columns

➢ Take 24 minutes to collect statistics for *all* tables and ***all*** columns.
  - Fast because all statistics are computed by integrating with Spark's built-in aggregate functions.
  - Should take much less time if we collect statistics for columns used in predicate, join, and group-by only.

SPARK SUMMIT EAST 2017

Preliminary Performance Test

- Query performance

Query	w/o CBO	w/ CBO	Speed up
Q8	28.8	22.9	1.3x
Q14a	3179.0	513.9	6.2x
Q14b	1769.5	479.3	3.7x
Q37	43.0	29.9	1.4x
Q60	179.5	169.3	1.1x
Q83	59.9	29.7	2.0x
etc	…..	…..	…..

➢ Good broadcast decision helps speed up

Current status

- SPARK-16026 is the umbrella jira.
  - A total of 24 sub-tasks have been created.
  - 17 sub-tasks have been resolved/closed.
  - 5 sub-tasks are coded and under review.
  - 2 sub-tasks are under development.
  - 5K+ lines of Scala code have been submitted.
- Expect to go in Spark 2.2.

Future work

- Advanced statistics: e.g. histograms, sketches.
- Partition level statistics.
- Provide detailed cost formula for each physical operator.
- Speed up statistics collection by sampling data for large tables.
- Etc.

Thank You.

ron.hu@huawei.com wangzhenhua@huawei.com

SPARK SUMMIT EAST 2017