Enhancing Spark SQL Optimizer with Reliable Statistics

Ron Hu, Fang Cao, Min Qiu*, Yizhen Liu

Huawei Technologies, Inc.

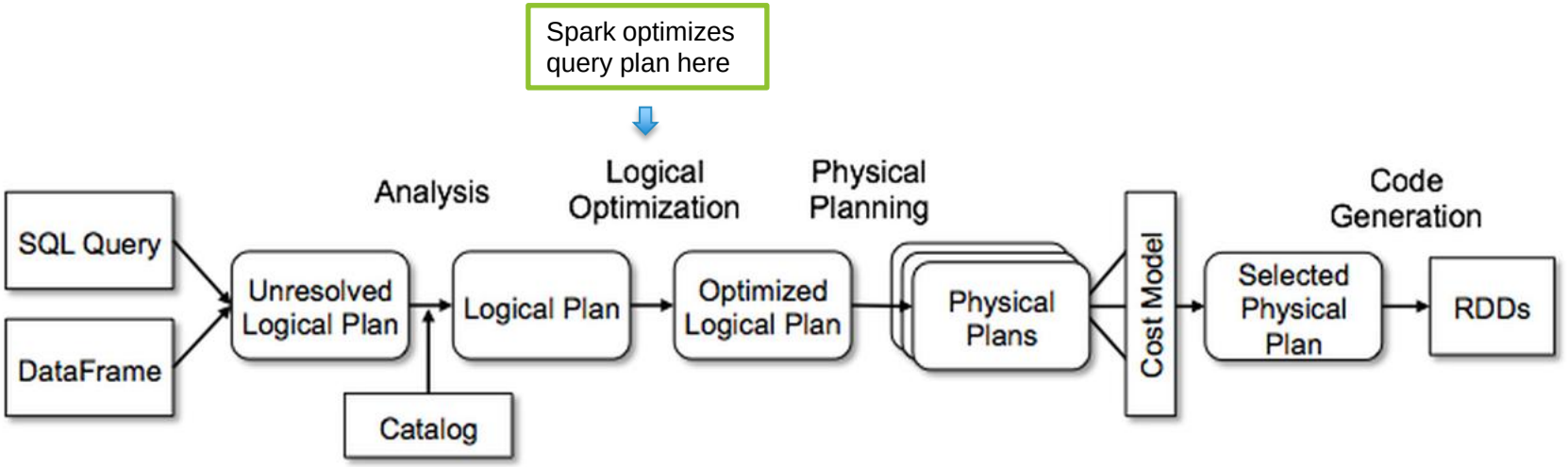
SPARK SUMMIT 2016
DATA SCIENCE AND ENGINEERING AT SCALE
JUNE 6-8, 2016 SAN FRANCISCO

* Former Huawei employee

Agenda

- Review of Catalyst Architecture
- Rule-based optimizations
- Reliable statistics collected
- Cost-based rules
- Future Work
- Q & A

Catalyst Architecture



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

Rule-Based Optimizer in Spark SQL

- Most of Spark SQL optimizer's rules are heuristics rules.
  - Does NOT consider the cost of each operator
  - Does NOT consider the cost of the equivalent logical plans
- Join order is decided by its position in the SQL queries
- Join type is based on some very simple system assumptions
- Number of shuffle partitions is a fixed number.
- Our community work:
  - Ex.: Fixed bugs in Spark.
  - Spark Summit East 2016 talk, https://spark-summit.org/east-2016/events/enhancements-on-spark-sql-optimizer/

Statistics Collected

- Collect Table Statistics information
- Collect Column Statistics information
- Only consider static system statistics (configuration file: CPU, Storage, Network) at this stage.
- Goal:
  - Calculate the cost for each database operator
    - in terms of number of output rows, size of output rows, etc.
  - Based on the cost calculation, adjust the query execution plan

Table Statistics Collected

- Use a modified Hive Analyze Table statement to collect statistics of a table.
  - Ex: `Analyze Table lineitem compute statistics`
- It collects table level statistics and save into metastore.
  - Number of rows
  - Number of files
  - Table size in bytes

Column Statistics Collected

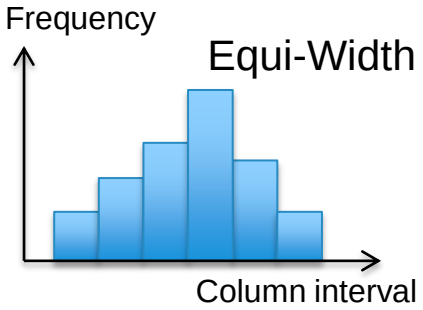
- Use Analyze statement to collect column level statistics of individual column.
  - Ex: `Analyze Table lineitem compute statistics for columns l_orderkey, l_partkey, l_suppkey, l_returnflag, l_linestatus, l_shipdate, ……..`
- It collects column level statistics and save into metastore.
  - Minimal value, maximal value,
  - Number of distinct values, number of null values
  - Column maximal length, column average length
  - Uniqueness of a column

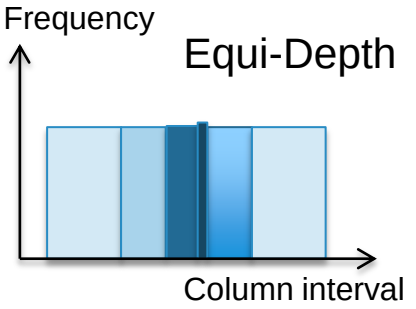
Column 1-D Histogram

Provided two kinds of Histograms: Equi-Width and Equi-Depth
- Between buckets, data distribution is determined by histograms
- Within one bucket, still assume data is evenly distributed

Max number of buckets: 256,
- If Number of Distinct Values <= 256, use equi-width
- If Number of Distinct Values > 256, use equi-depth

Used Hive Analyze Command and Hive Metastore API





Column 2-D Histogram

- Developed 2-dimensional equi-depth histogram for the column combination of (c1, c2)
  - In a 2-dimensional histogram, there are 2 levels of buckets.
  - B(c1) is the number of major buckets for column C1.
  - Within each C1 bucket, B(c2) is the number of buckets for C2

- Lessons Learned:
  - Users do not use 2-D histogram often as they do not know which 2 columns are correlated.
  - What granularity to use? 256 buckets or 256x256 buckets?
  - Difficult to extend to 3-D or more dimensions
  - Can be replaced by hints

Cost-Based Rules

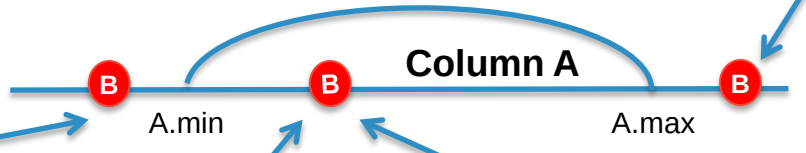
- Optimizer is a RuleExecutor.
  - Individual optimization is defined as Rule
- We added new rules to estimate number of output rows and output size in bytes for each execution operator:
  - MetastoreRelation, *Filter*, Project, *Join*, Sort, Aggregate, Exchange, Limit, Union, etc.
- The node's cost = nominal scale of (output_rows, output_size)

Filter Operator Statistics

- Between Filter's expressions: AND, OR, NOT
- In each Expression: =, <, <=, >, >=, like, in, etc
- Current support type in Expression
  - For <, <=, >, >=, String, Integer, Double, etc
  - For =, String, Integer, Double and Date Type, and User-Defined Types, etc.
- Sample: A <= B
  - Based on A, B's min/max/NDV values, decide the relationships between A and B. After completing this expression, what the new min/max/NDV should be for A and B
  - We use histograms to adjust min/max/NDV values
  - Assume all the data is evenly distributed if no histogram information.

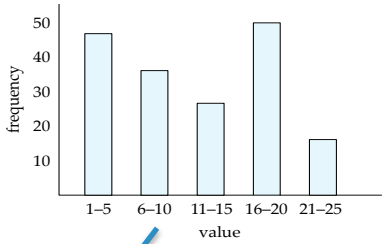
Filter Operator Example

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



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

Filtering Factor = 0%
no need to change A’s statistics
A will not appear in the future work



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*

With Histograms
Filtering Factor = using Histograms to calculate
A.min = no change
A.max = B.value
*A.ndv = A.ndv * Filtering Factor*

Filter Operator Example

- ***Column A (op) Column B***
  - Actually, based on observation, this expression will appear in Project, but not in Filter
  - Note: for column comparing, currently we don't support histogram. We cannot suppose the data is evenly distributed, so the empirical filtering factor is set to **1/3**
  - (op) can be "<", "<=", ">", ">="
  - Need to adjust the A and B's min/max/NDV after filtering
  - Sample: **Column A < Column B**

A filtering = 100%
B filtering = 100%

A filtering = 0%
B filtering = 0%

A filtering = 33.3%
B filtering = 33.3%

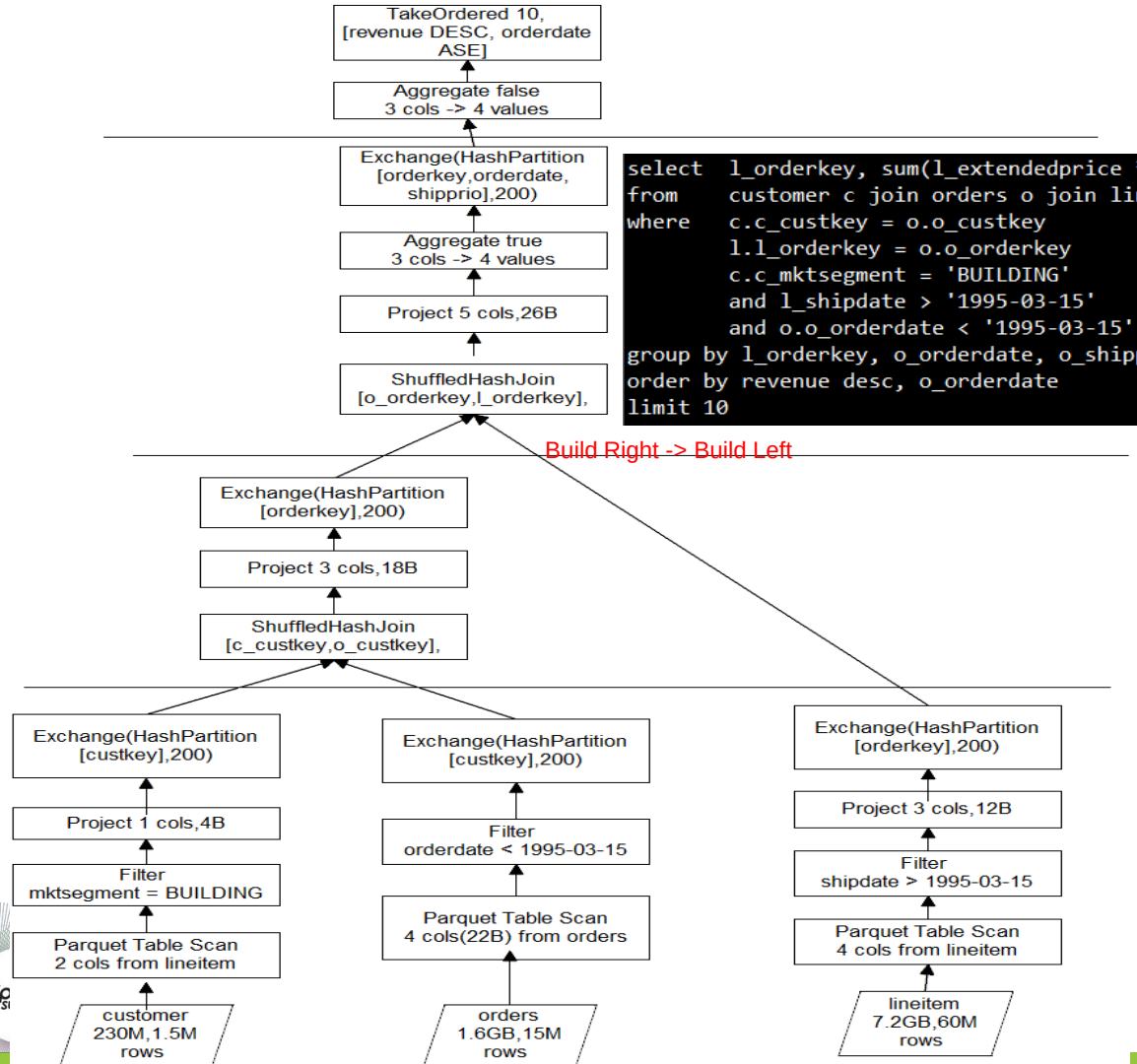
A filtering = 33.3%
B filtering = 33.3%

Join Order

- Only for two table joins
- We calculate the cost of Hash Join using the stats of left and right nodes.
  - Nominal Cost = <nominal-rows> × 0.7 + <nominal-size> × 0.3
- Choose lower-cost child as build side of hash join (Prior to Spark 1.5).

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 table.
  - The join with smaller output is performed first.
  - Can benefit star join queries (like TPC-DS).
- Using dynamic programming for join order




```
select  l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriori
from    customer c join orders o join lineitem l
where   c.c_custkey = o.o_custkey
        l.l_orderkey = o.o_orderkey
        c.c_mktsegment = 'BUILDING'
        and l_shipdate > '1995-03-15'
        and o.o_orderdate < '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
limit 10
```

Sample：Q3，3 Tables join+aggregate

- ParquetRelation node
- Filter node
- Project node
- Join node
- Aggregation
- Limit

Limitation without Key Information

- Spark SQL does not support index or primary key.
  - This missing information fails to properly estimate the join output of the primary/foreign key join.
- When estimating the number of GROUP BY operator output records, we multiply the number of distinct values for each GROUP BY column.
  - This formula is valid only if every GROUP BY column is independent.

Column Uniqueness

- We know that a column is unique (or primary key) if the number of distinct values divided by the number of records of a table is close to 1.0.
  - We can set the size of hash join table properly if one join column is unique.
  - When computing the number of GROUP BY output records, if one GROUP BY column is unique, we do NOT multiply those non-unique columns.

Unique Column Example, tpc-h Q10

```
/* tpc-h Q10: c_custkey is unique */

SELECT c_custkey, c_name, sum(l_extendedprice * (1 - l_discount))
     AS revenue, c_acctbal, n_name, c_address, c_phone, c_comment
  FROM nation join customer join orders join lineitem
 WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey
   AND o_orderdate >= '1993-10-01' AND o_orderdate < '1994-01-01'
   AND l_returnflag = 'R' AND c_nationkey =  n_nationkey
 GROUP BY c_custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment
 ORDER BY revenue DESC limit 20
```

Number of group-by outputs can be:

- 1708M if there is no unique column information,
- 82K if we know there is a unique group-by column

SQL Hints

- Some information cannot be analyzed directly from the statistics of tables/columns. Example, tpc-h Q13:

```
SELECT c_count, count(*) as custdist
  FROM
  (SELECT c_custkey, count(o_orderkey) c_count
     FROM customer LEFT OUTER JOIN orders
       ON c_custkey = o_custkey
      and o_comment not like '%special%request%'
    GROUP BY c_custkey
  ) c_orders
 GROUP BY c_count
 ORDER BY custdist desc, c_count desc
```

  - Supported hints `/*+ …. */`: `Like_FilterFactor, NDV_Correlated_Columns, Join_Build, Join_Type, ……`

Actual vs Estimated Output Rows

Query	Actual	Estimated
Q1	4	6
Q2	460	1756
Q3	11621	496485
Q4	5	5
Q5	5	25
Q6	1	1
Q7	4	5
Q8	2	5
Q9	175	222
Q10	37967	81611

Query	Actual	Estimated
Q11	28574	32000
Q12	2	2
Q13	42	100
Q14	1	1
Q15	1	2
Q16	18314	14700
Q17	1	1
Q18	57	1621
Q19	1	1
Q20	186	558
Q21	411	558

Wrong Output Rows Estimate for Q3

- We do not handle the correlated columns of different tables.

```
TPC-H Q3:

select l_orderkey, sum(l_extendedprice *(1 - l_discount)) as revenue,
       o_orderdate, o_shippriority
  from customer, orders, lineitem
 where c_mktsegment = 'BUILDING'
   and c_custkey = o_custkey and l_orderkey = o_orderkey
   and o_orderdate < date '1995-3-15'
   and l_shipdate > date '1995-3-15'
 group by l_orderkey, o_orderdate, o_shippriority
 order by l_orderkey, revenue desc, o_orderdate
```

Possible Future Work

- How to collect table histograms information quickly and correctly
  - For full table scan – correct, but slow, especially for big data
  - Possible method – Sampling Counting
    - Linear, LogLog, Adaptive, Hyper LogLog, Hyper LogLog++, etc

- Expression Statistics
  - Now only raw columns' statistics are collected. Not for the derived columns
  - Derived columns from calculation of expressions
    - Ex: Alias Column, Aggregation Expression, Arithmetic Expression, UDF

- Collecting the real-world running statistics information, for the future query plan optimization.
  - Continuous feedback optimization

THANK YOU.

ron.hu@huawei.com fang.cao@huawei.com
yizhen.liu@huawei.com