HUAWEI

Cost-Based Optimizer Framework for Spark SQL

王振华

LEADING NEW ICT

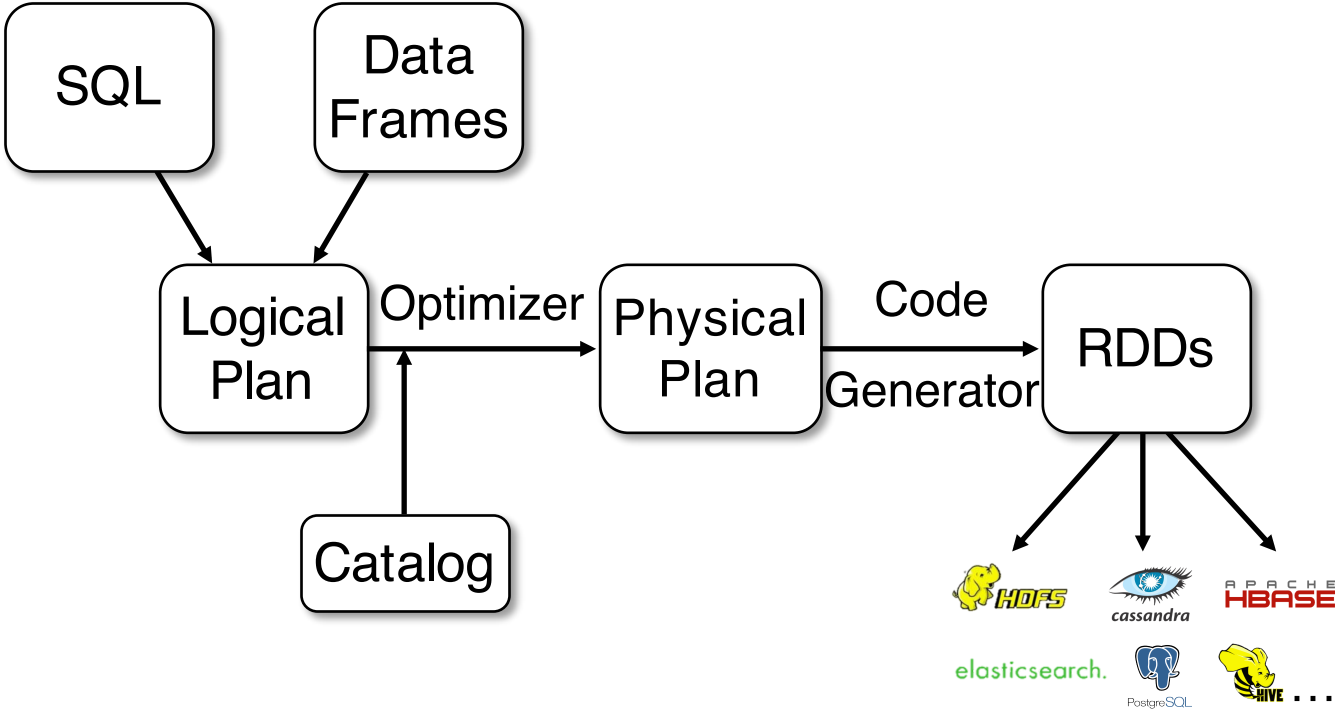
Overview

- Motivation
- Statistics Collection Framework
- Cost Based Optimizations
- TPC-DS Benchmark and Query Analysis
- Current Status and Future Work

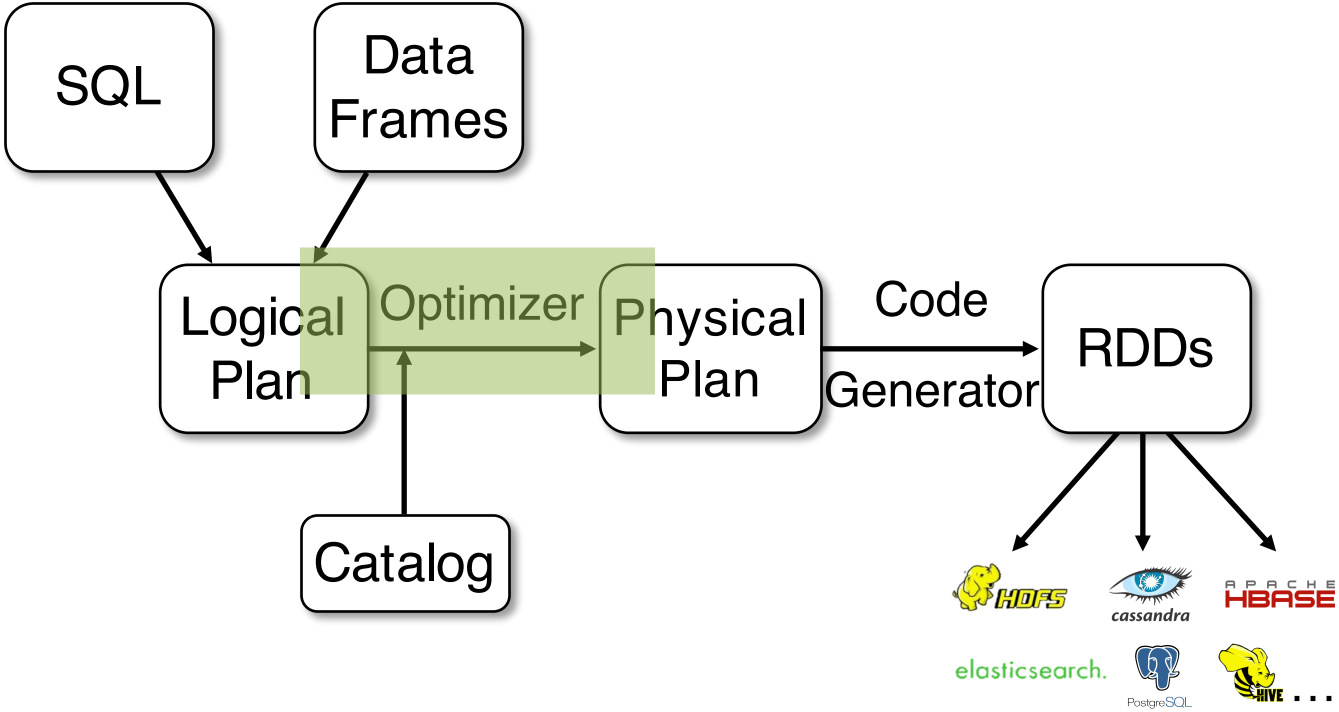
Overview

- Motivation
- Statistics Collection Framework
- Cost Based Optimizations
- TPC-DS Benchmark and Query Analysis
- Current Status and Future Work

How Spark Executes a Query?



How Spark Executes a Query?

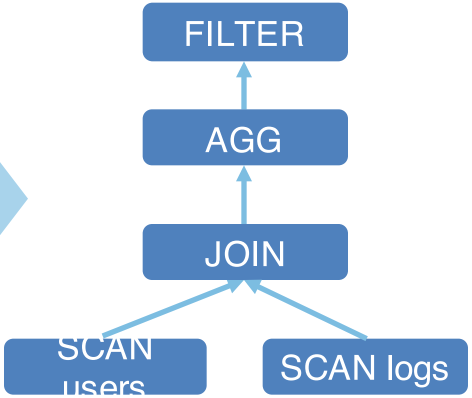


Catalyst Optimizer: An Overview

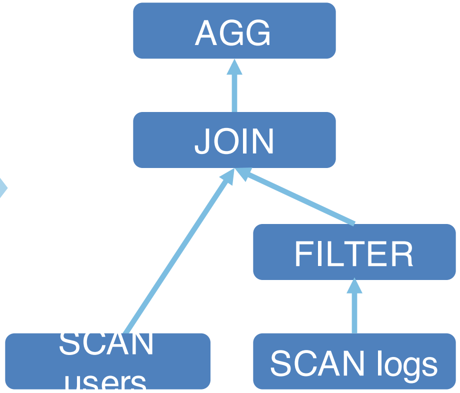
```
events =
 sc.read.json("/logs")

stats =
  events.join(users)
  .groupBy("loc","status")
.avg("duration")


errors = stats.where(
  stats.status == "ERR")
```



Query Plan is an internal representation of a user's program

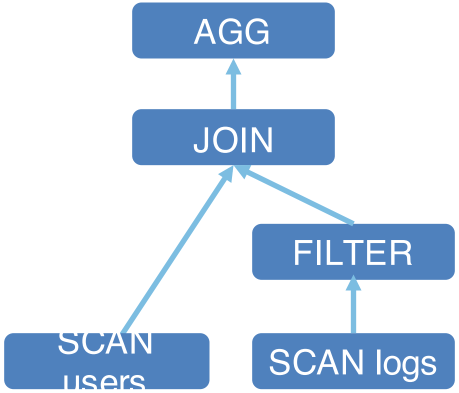


Series of Transformations that convert the initial query plan into an optimized plan

Catalyst Optimizer: An Overview

In Spark, the optimizer's goal is to minimize end-to-end query response time. Two key ideas:

- Prune unnecessary data as early as possible
    - e.g., filter pushdown, column pruning
- Minimize per-operator cost
    - e.g., broadcast vs shuffle

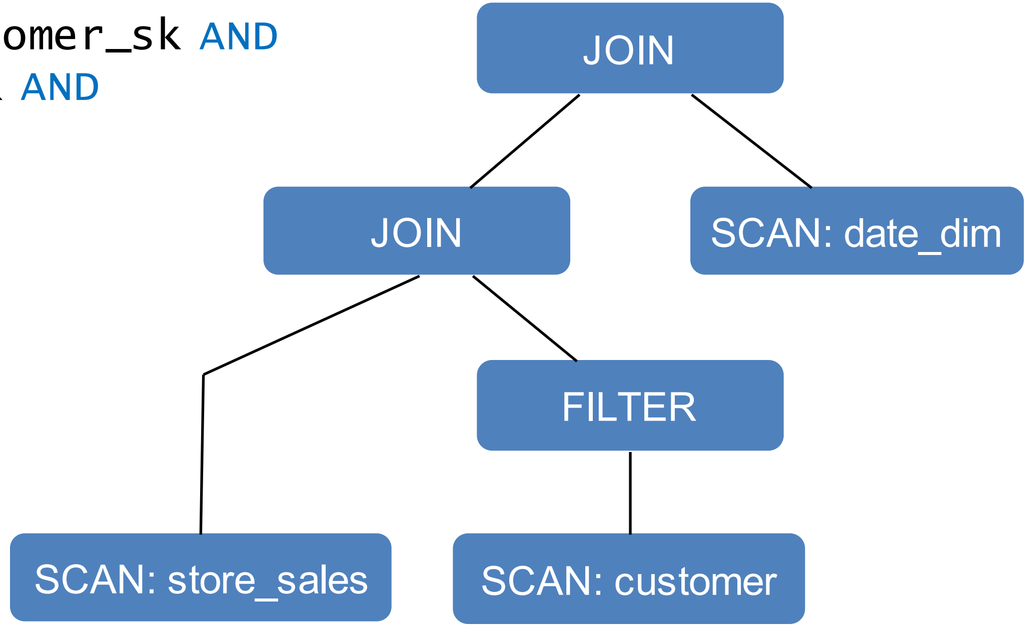


Rule-based Optimizer in Spark 2.1

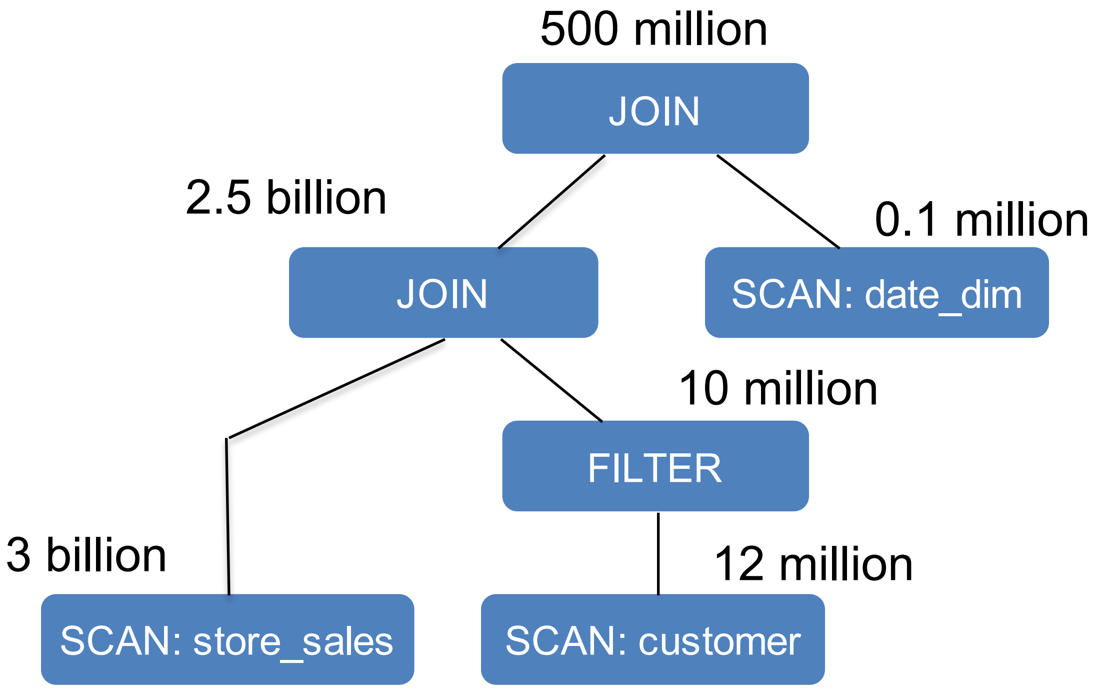
- Most of Spark SQL optimizer's rules are heuristics rules.
  - `PushDownPredicate`, `ColumnPruning`, `ConstantFolding`,…
- Does NOT consider the cost of each operator
- Does NOT consider selectivity when estimating join relation size
- Join order is mostly decided by its position in the SQL queries
- Physical Join implementation is decided based on heuristics

An Example (TPC-DS q11 variant)

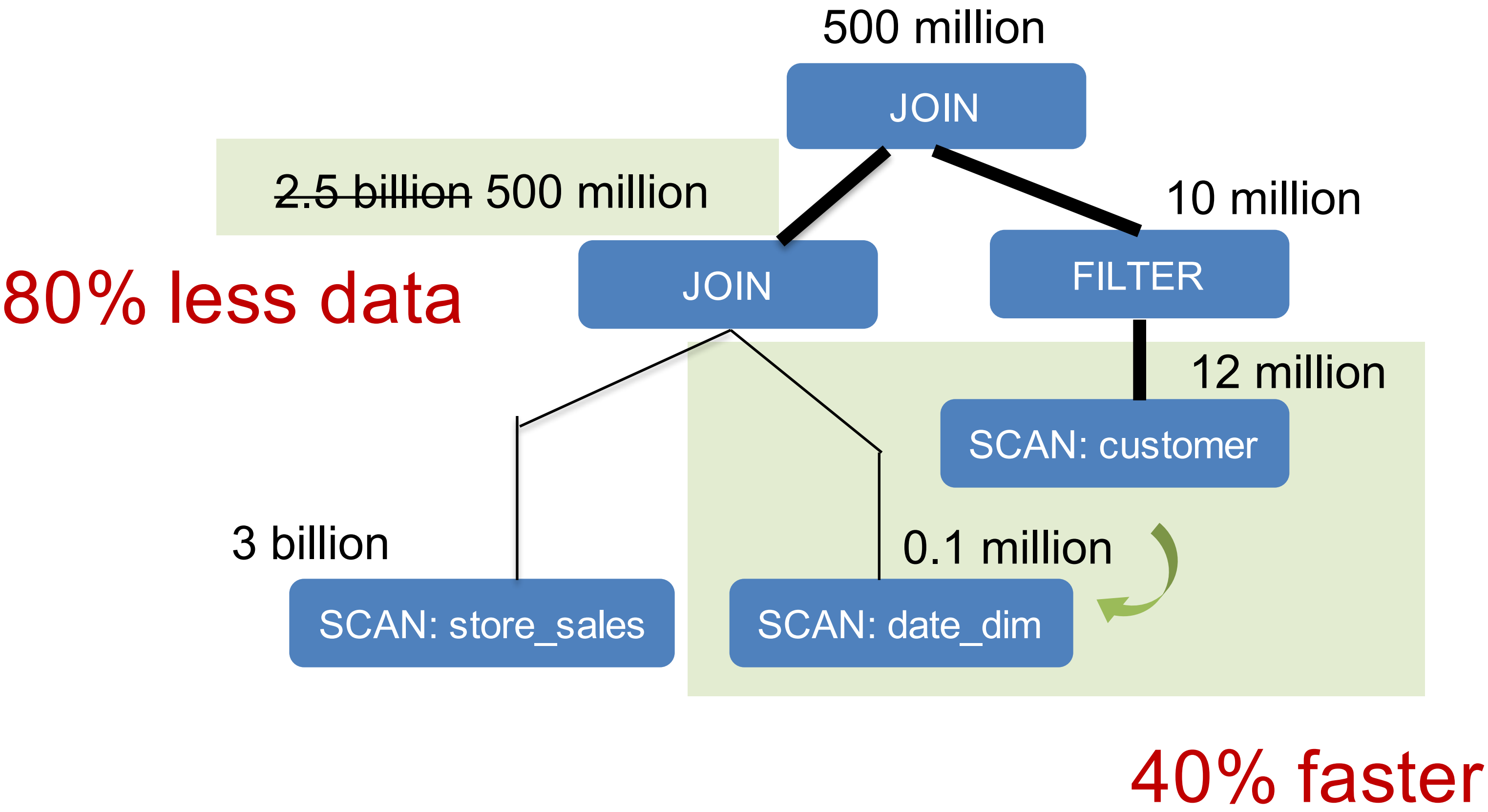
```
SELECT customer_id
FROM customer, store_sales, date_dim
WHERE c_customer_sk = ss_customer_sk AND
  ss_sold_date_sk = d_date_sk AND
  c_customer_sk > 1000
```



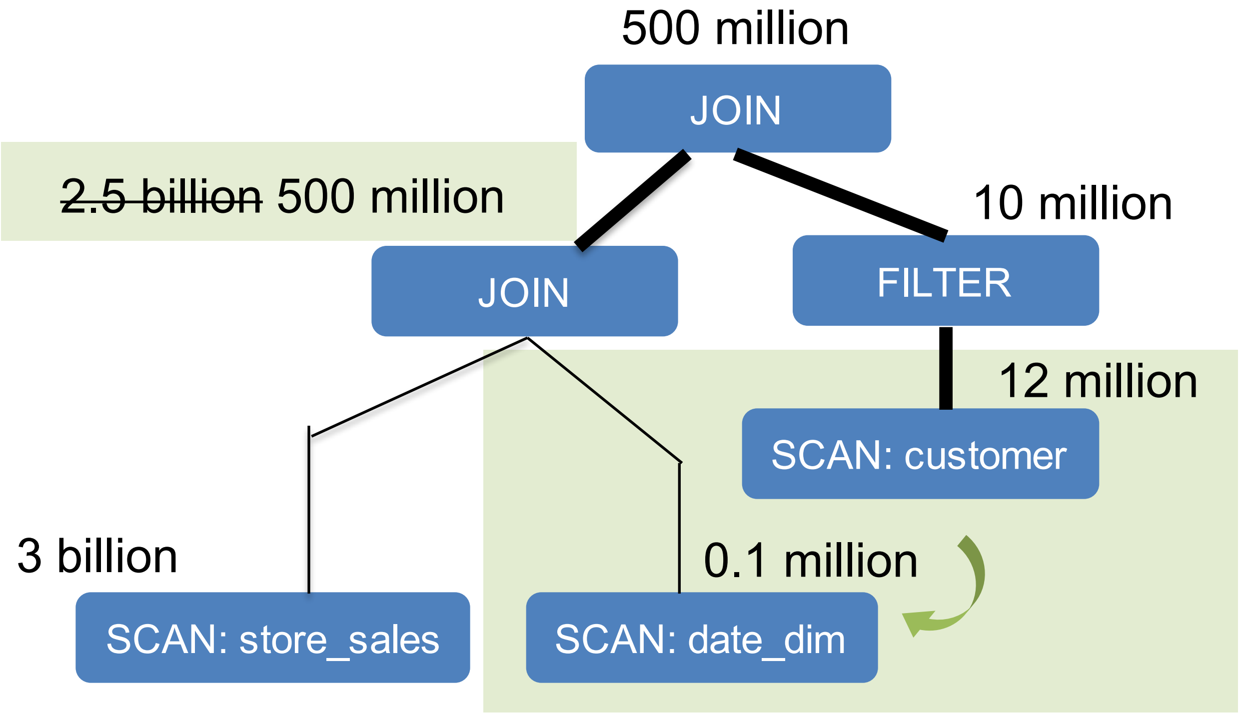
An Example (TPC-DS q11 variant)



An Example (TPC-DS q11 variant)



An Example (TPC-DS q11 variant)



How do we automatically optimize queries like these?

Cost Based Optimizer (CBO)

- Collect, infer and propagate table/column statistics on source/intermediate data
- Calculate the cost for each operator in terms of number of output rows, size of output, etc.
- Based on the cost calculation, pick the most optimal query execution plan

Overview

- Motivation
- Statistics Collection Framework
- Cost Based Optimizations
- TPC-DS Benchmark and Query Analysis
- Current Status and Future Work

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

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

- In each logical expression: =, <, <=, >, >=, in, etc
- Combinations between Logical expressions: AND, OR, NOT
- 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.

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**

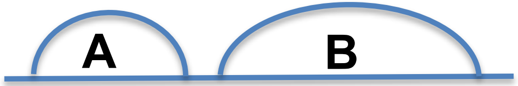
Selectivity = 100%
no need to change A’s statistics

Selectivity = 0%
need to change A’s statistics

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

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**



Selectivity = 100%

Selectivity = 0%

Selectivity = 33.3%

Selectivity = 33.3%

Join Cardinality Estimation

- *Inner-Join:* The number of rows of "A join B on A.k1 = B.k1" is estimated as:
- `num`(A⋈B) = `num`(A) * `num`(B) / `max(distinct`(A.k1), `distinct`(B.k1)),
  - where `num`(A) is the number of records in table A, `distinct` is the number of distinct values of that column.
  - The underlying assumption for this formula is that each value of the smaller domain is included in the larger domain.

- We similarly estimate cardinalities for Left-Outer Join, Right-Outer Join and Full-Outer Join

Other Operator Estimation

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

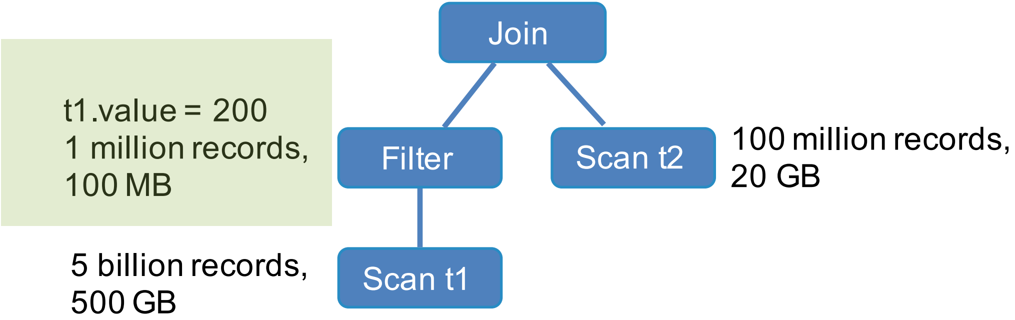
Overview

- Motivation
- Statistics Collection Framework
- Cost Based Optimizations
- TPC-DS Benchmark and Query Analysis
- Current Status and Future Work

Build Side Selection

- For two-way hash joins, we need to choose one operand as build side and the other as probe side.

- Choose lower-cost child as build side of hash join.
  - Without CBO: build side was selected based on original table sizes. ⇨ BuildRight
  - With CBO: build side is selected based on estimated cost of various operators before join. ⇨ BuildLeft



HUAWEI

Hash Join Implementation: Broadcast vs. Shuffle

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

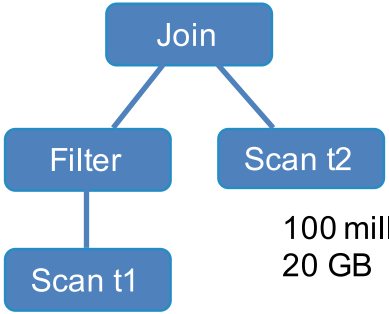
Logical Plan

- ➢ Equi-join
  - Inner Join
  - LeftSemi/LeftAnti Join
  - LeftOuter/RightOuter Join
- ➢ Theta-join

Physical Plan

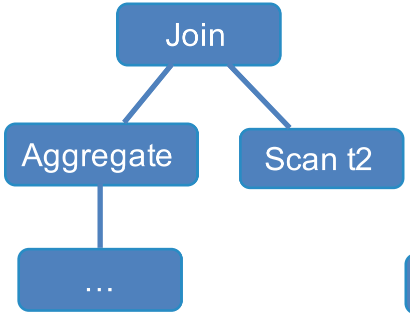
- ➢ SortMergeJoinExec/
  BroadcastHashJoinExec/
  ShuffledHashJoinExec
- ➢ CartesianProductExec/
  BroadcastNestedLoopJoinExec

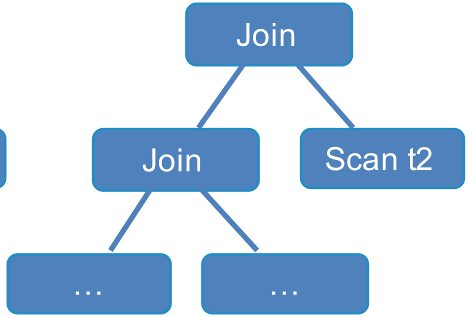
t1.value = 100
Only 1000 records,
100 KB



5 billion records,
500 GB

100 million records,
20 GB





HUAWEI

Multi-way Join Reorder

- Reorder the joins using a dynamic programming algorithm.
  1. First we put all items (basic joined nodes) into level 0.
  2. Build all two-way joins at level 1 from plans at level 0 (single items).
  3. Build all 3-way joins from plans at previous levels (two-way joins and single items).
  4. Build all 4-way joins etc, until we build all n-way joins and pick the best plan among them.

- When building m-way joins, only keep the best plan (optimal sub-solution) for the same set of m items.
  - E.g., for 3-way joins of items {A, B, C}, we keep only the best plan among: `(A J B) J C, (A J C) J B and (B J C) J A`

Multi-way Join Reorder

Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one

Selinger et al. Access Path Selection in a Relational Database Management System. In SIGMOD 1979

Join Cost Formula

- The cost of a plan is the sum of costs of all intermediate tables.

- $\texttt{Cost} = \texttt{weight} \times \texttt{Cost}_{cpu} + \texttt{Cost}_{IO} \times (1 - \texttt{weight})$
  - In Spark, we use

    `weight * cardinality + size * (1 – weight)`
  - `weight` is a tuning parameter configured via `spark.sql.cbo.joinReorder.card.weight` (0.7 as default)

Overview

- Motivation
- Statistics Collection Framework
- Cost Based Optimizations
- TPC-DS Benchmark and Query Analysis
- Current Status and Future Work

Preliminary Performance Test

- Setup:
  - TPC-DS size at 1 TB (scale factor 1000)
  - 4 node cluster (Huawei FusionServer RH2288: 40 cores, 384GB mem)
  - Apache Spark 2.2 RC (dated 5/12/2017)

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

➢ Take 14 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.

TPC-DS Query Q11

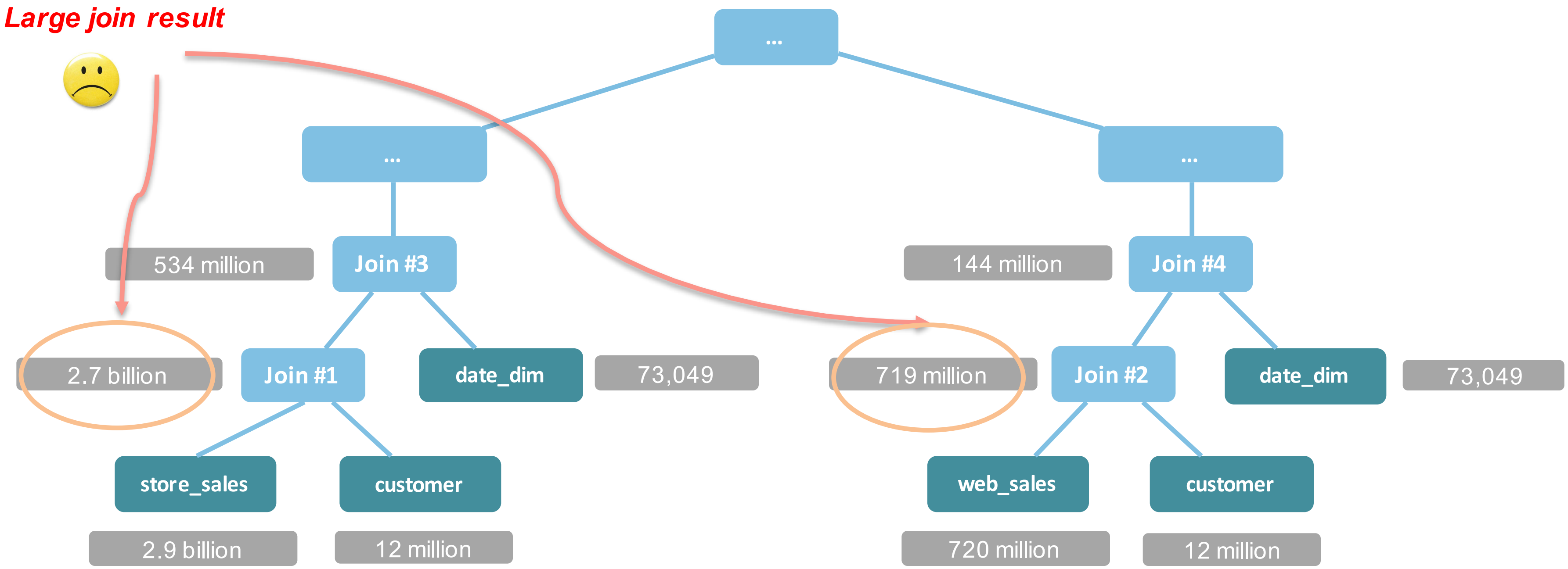
```
WITH year_total  AS (
 SELECT
   c_customer_id  customer_id,
   c_first_name  customer_first_name,
   c_last_name  customer_last_name,
   c_preferred_cust_flag   customer_preferred_cust_fl ag,
   c_birth_country  customer_birth_country,
   c_login  customer_login,
   c_email_address  customer_email_address,
   d_year  dyear,
   sum(ss_ext_list_price  - ss_ext_discount_amt)   year_total,
   's' sale_type
 FROM customer, store_sales,  date_dim
 WHERE c_customer_sk  = ss_customer_sk
   AND ss_sold_date_sk   = d_date_sk
 GROUP BY c_customer_id,  c_first_name,  c_last_name,  d_year
   , c_preferred_cust_flag,   c_birth_country,  c_login,  c_email_address,   d_year
 UNION ALL
 SELECT
   c_customer_id  customer_id,
   c_first_name  customer_first_name,
   c_last_name  customer_last_name,
   c_preferred_cust_flag   customer_preferred_cust_fl ag,
   c_birth_country  customer_birth_country,
   c_login  customer_login,
   c_email_address  customer_email_address,
   d_year  dyear,
   sum(ws_ext_list_price  - ws_ext_discount_amt)   year_total,
   'w' sale_type
 FROM customer, web_sales,  date_dim
 WHERE c_customer_sk  = ws_bill_customer_sk  AND ws_sold_date_sk   = d_date_sk
 GROUP BY c_customer_id,  c_first_name,  c_last_name,  c_preferred_cust_flag,
   c_birth_country,   c_login,  c_email_address,   d_year)
```



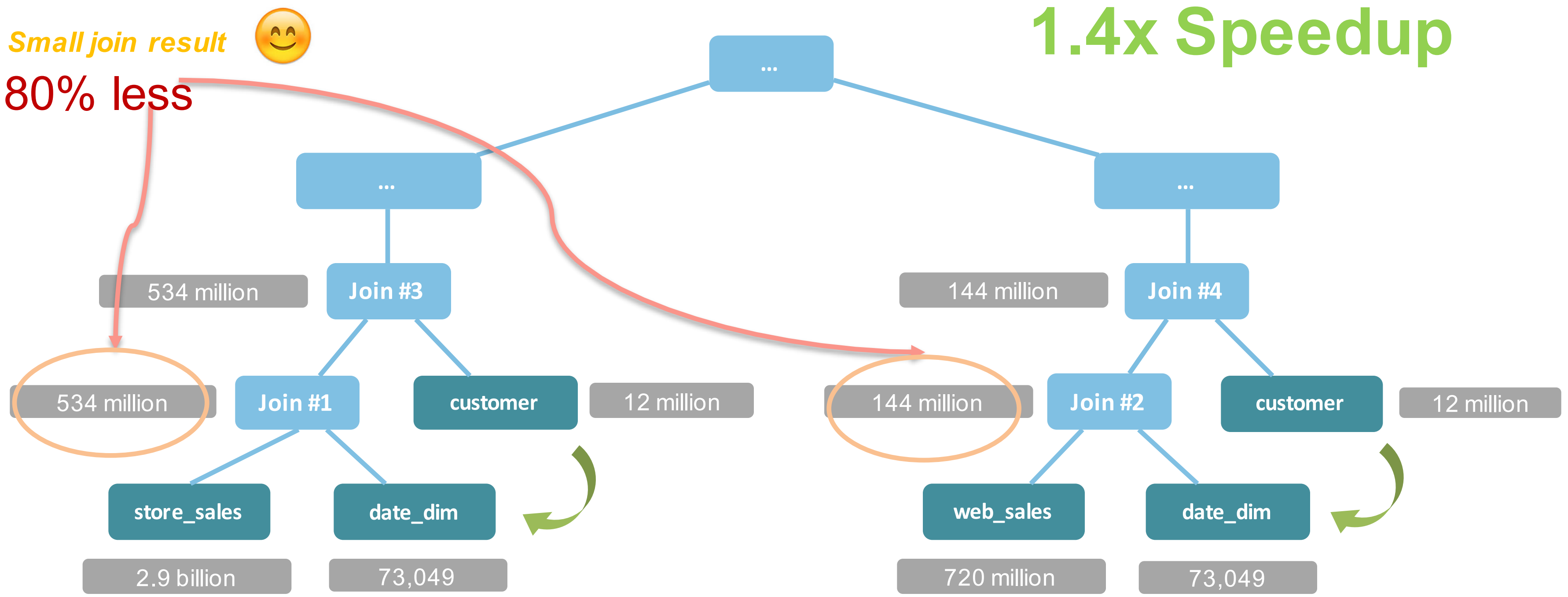
```
SELECT t_s_secyear.customer_preferred_c ust_flag
FROM year_total  t_s_firstyear
 , year_total  t_s_secyear
 , year_total  t_w_firstyear
 , year_total  t_w_secyear
WHERE t_s_secyear.customer_id   = t_s_firstyear.customer_id
 AND t_s_firstyear.customer_id  = t_w_secyear.customer_id
 AND t_s_firstyear.customer_id  = t_w_firstyear.customer_id
 AND t_s_firstyear.sale_type   = 's'
 AND t_w_firstyear.sale_type   = 'w'
 AND t_s_secyear.sale_type   = 's'
 AND t_w_secyear.sale_type   = 'w'
 AND t_s_firstyear.dyear  = 2001
 AND t_s_secyear.dyear  = 2001  + 1
 AND t_w_firstyear.dyear  = 2001
 AND t_w_secyear.dyear  = 2001  + 1
 AND t_s_firstyear.year_total   > 0
 AND t_w_firstyear.year_total   > 0
 AND CASE WHEN t_w_firstyear.year_total   > 0
 THEN t_w_secyear.year_total   / t_w_firstyear.year_total
    ELSE NULL END
 > CASE WHEN t_s_firstyear.year_total   > 0
 THEN t_s_secyear.year_total   / t_s_firstyear.year_total
   ELSE  NULL END
ORDER BY t_s_secyear.customer_preferred_cus t_fl ag
LIMIT 100
```

HUAWEI

Query Analysis – Q11 CBO OFF



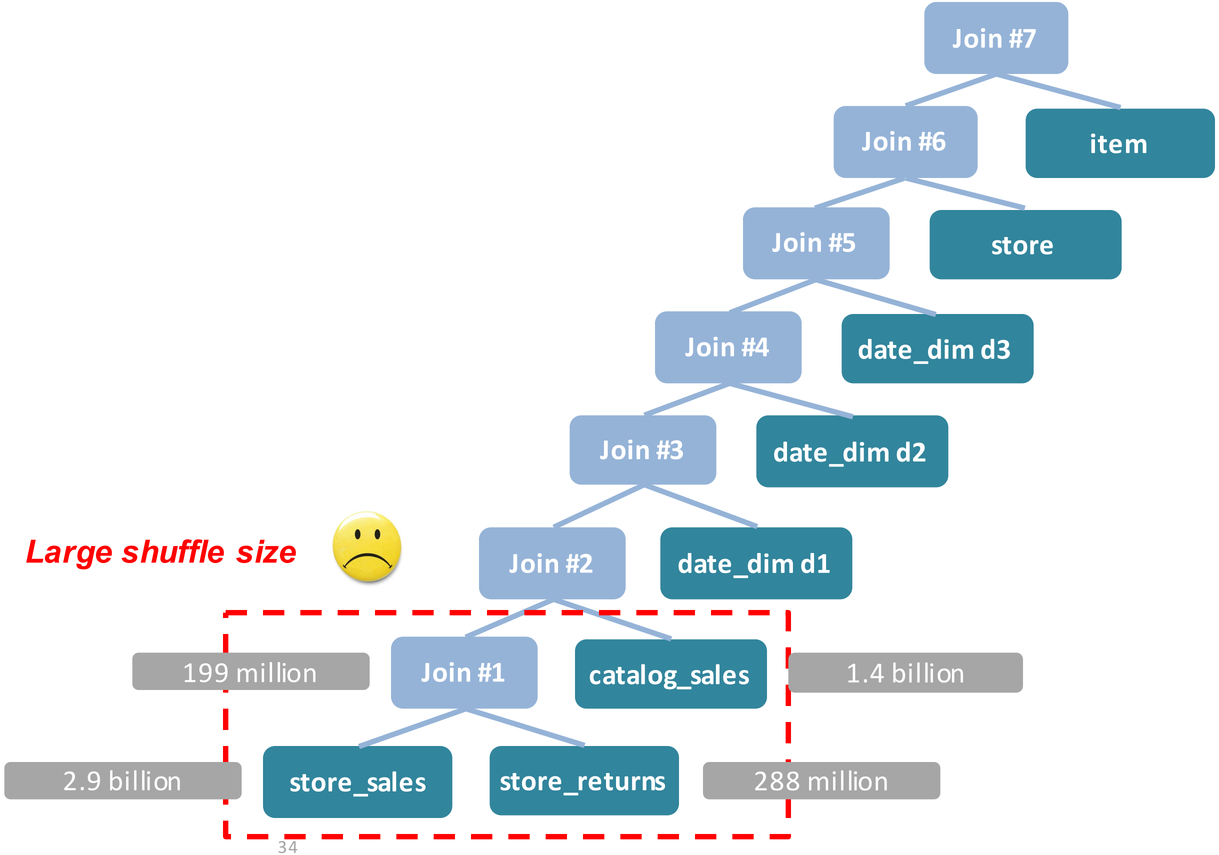
Query Analysis – Q11 CBO ON



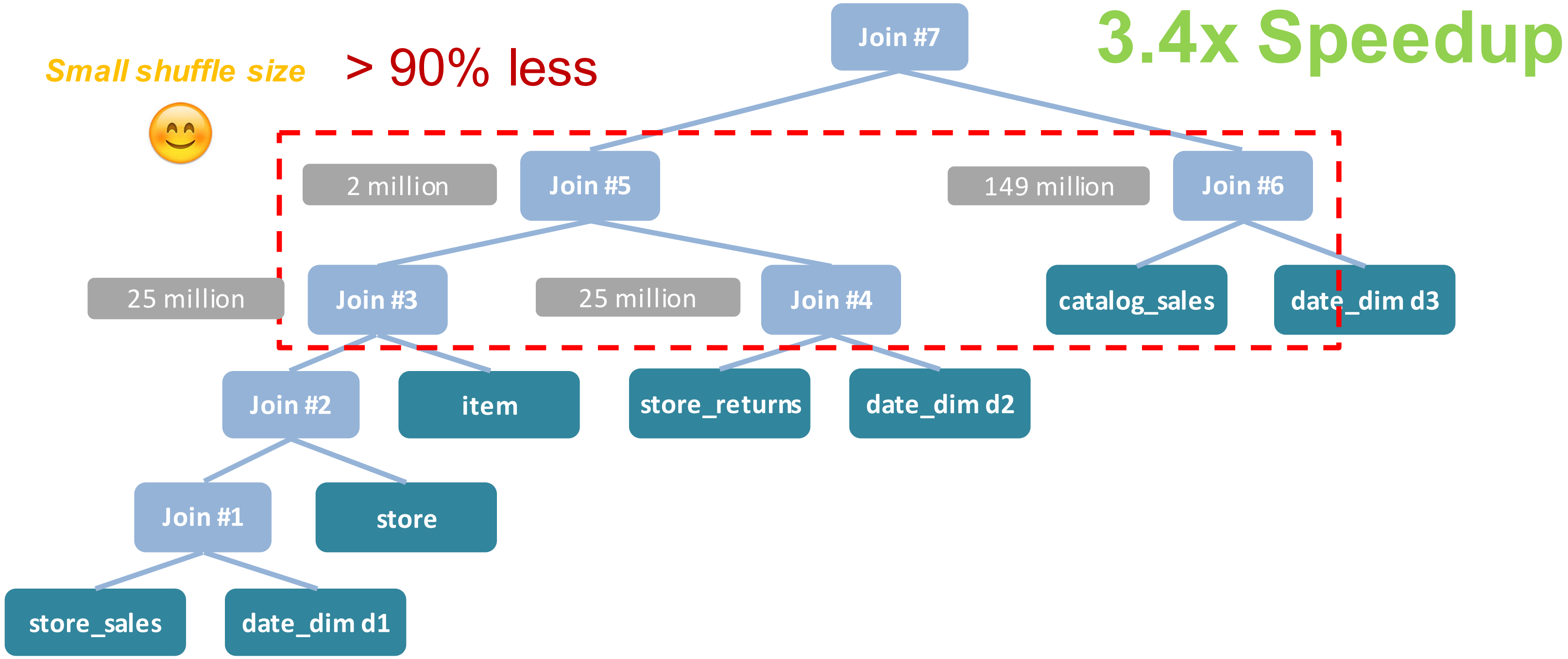
TPC-DS Query 25

```sql
SELECT i_item_id, i_item_desc, s_store_id, s_store_name,
  sum(ss_net_profit) AS store_sales_profit,
  sum(sr_net_loss) AS store_returns_loss,
  sum(cs_net_profit) AS catalog_sales_profit
FROM store_sales, store_returns, catalog_sales,
  date_dim d1, date_dim d2, date_dim d3, store, item
WHERE d1.d_moy = 4
    AND d1.d_year = 2001
    AND d1.d_date_sk = ss_sold_date_sk
    AND i_item_sk = ss_item_sk
    AND s_store_sk = ss_store_sk
    AND ss_customer_sk = sr_customer_sk
    AND ss_item_sk = sr_item_sk
    AND ss_ticket_number = sr_ticket_number
    AND sr_returned_date_sk = d2.d_date_sk
    AND d2.d_moy BETWEEN 4 AND 10
    AND d2.d_year = 2001
    AND sr_customer_sk = cs_bill_customer_sk
    AND sr_item_sk = cs_item_sk
    AND cs_sold_date_sk = d3.d_date_sk
    AND d3.d_moy BETWEEN 4 AND 10
    AND d3.d_year = 2001
GROUP BY i_item_id, i_item_desc, s_store_id, s_store_name
ORDER BY i_item_id, i_item_desc, s_store_id, s_store_name
LIMIT 100
```

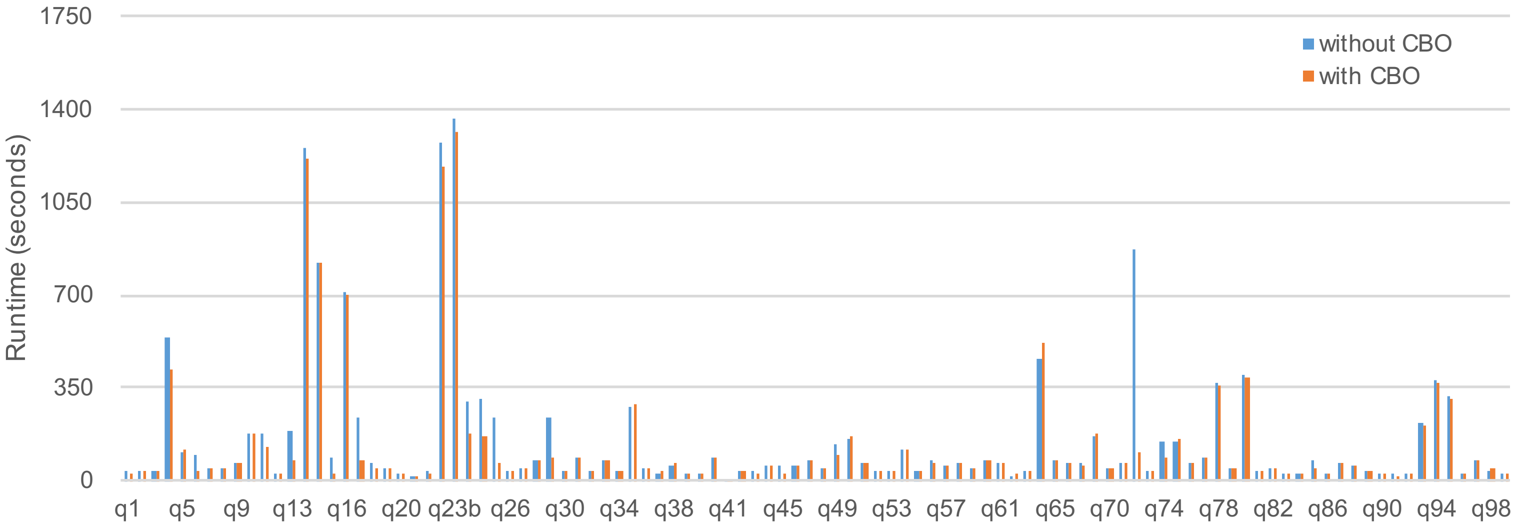
Query Analysis – Q25 CBO OFF



Query Analysis – Q25 CBO ON



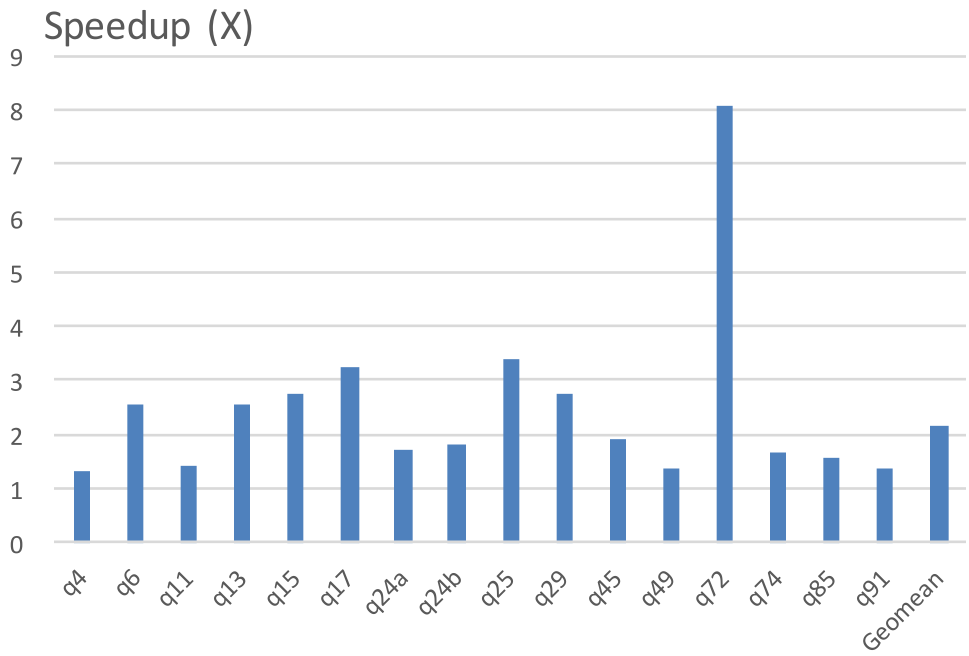
TPC-DS Query Performance



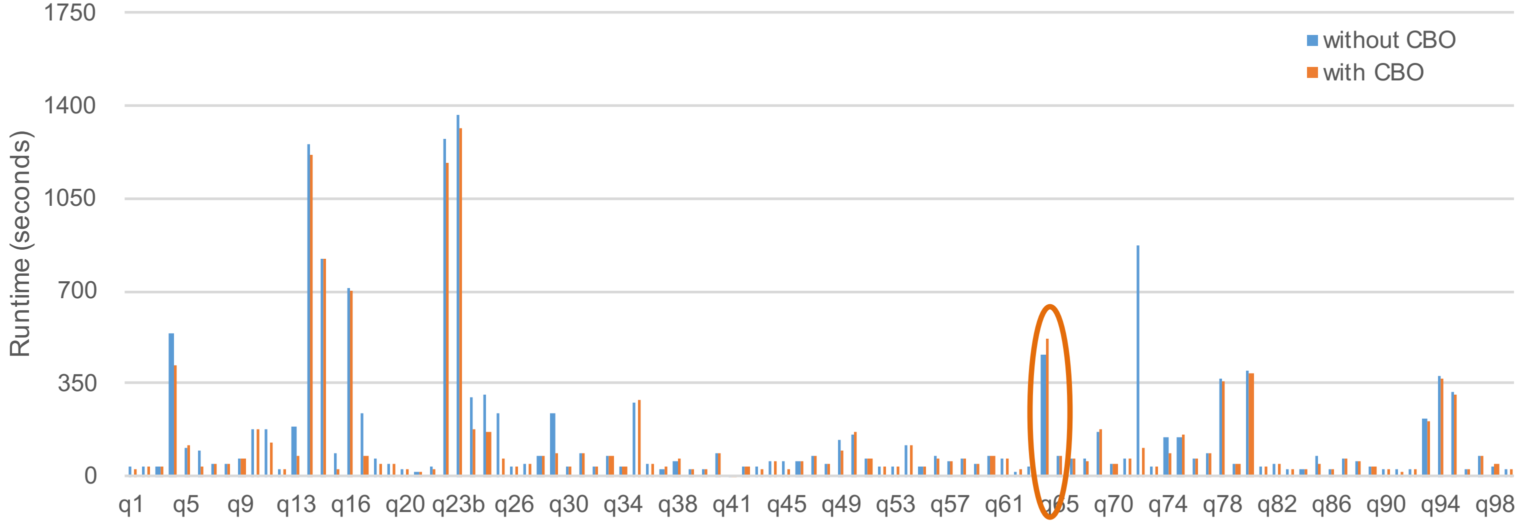
TPC-DS Query Speedup

- TPC-DS query speedup ratio with CBO versus without CBO

- 16 queries show speedup > 30%
- The max speedup is 8X.
- The geo-mean of speedup is 2.2X.



TPC-DS Query Performance

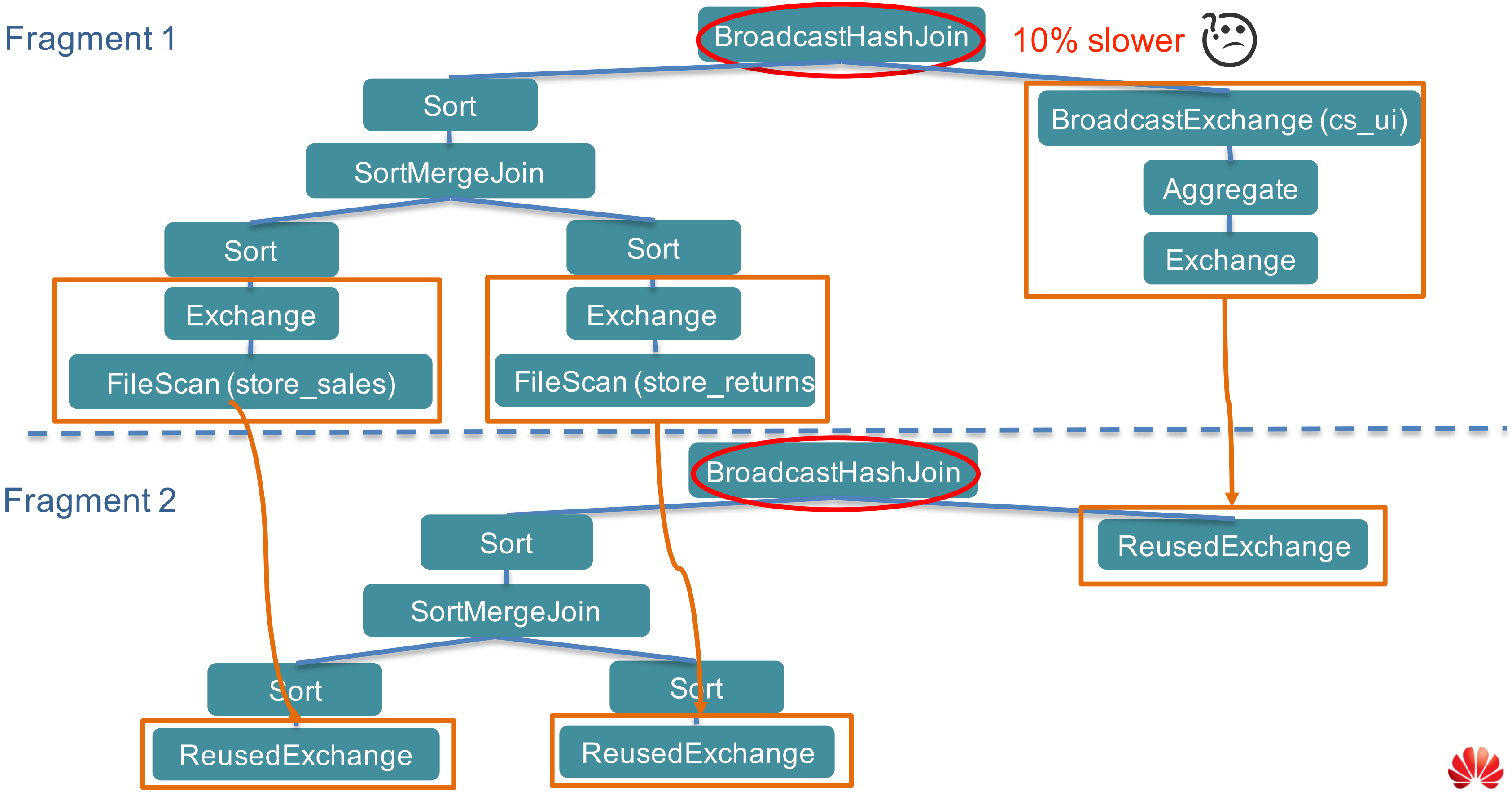


TPC-DS Query 64

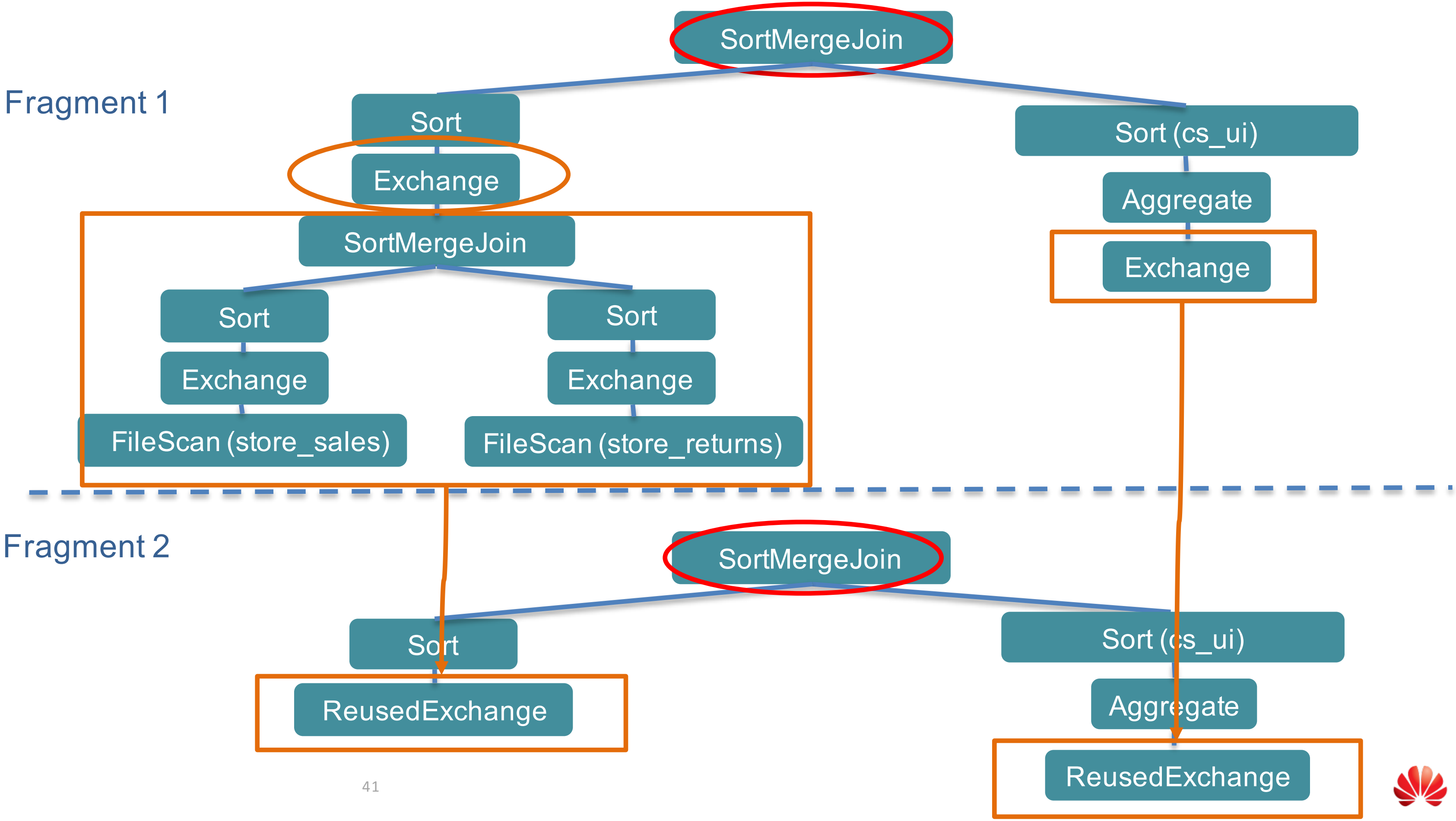
```sql
WITH cs_ui AS
 (SELECT
    cs_item_sk,
    sum(cs_ext_list_price) AS sale,
    sum(cr_refunded_cash + cr_reversed_charge + cr_store_credit) AS refund
  FROM catalog_sales, catalog_returns
  WHERE cs_item_sk = cr_item_sk AND cs_order_number = cr_order_number
  GROUP BY cs_item_sk
  HAVING sum(cs_ext_list_price) > 2 * sum(cr_refunded_cash + cr_reversed_charge + cr_store_credit)),
    cross_sales AS
  (SELECT
    i_product_name product_name, i_item_sk item_sk, s_store_name store_name,
    s_zip store_zip, ad1.ca_street_number b_street_number, ad1.ca_street_name b_streen_name,
    ad1.ca_city b_city, ad1.ca_zip b_zip, ad2.ca_street_number c_street_number,
    ad2.ca_street_name c_street_name, ad2.ca_city c_city, ad2.ca_zip c_zip,
    d1.d_year AS syear, d2.d_year AS fsyear, d3.d_year s2year,
    count(*) cnt, sum(ss_wholesale_cost) s1, sum(ss_list_price) s2, sum(ss_coupon_amt) s3
  FROM store_sales, store_returns, cs_ui, date_dim d1, date_dim d2, date_dim d3,
    store, customer, customer_demographics cd1, customer_demographics cd2,
    promotion, household_demographics hd1, household_demographics hd2,
    customer_address ad1, customer_address ad2, income_band ib1, income_band ib2, item
  WHERE ss_store_sk = s_store_sk AND ss_sold_date_sk = d1.d_date_sk AND
    ss_customer_sk = c_customer_sk AND ss_cdemo_sk = cd1.cd_demo_sk AND
    ss_hdemo_sk = hd1.hd_demo_sk AND ss_addr_sk = ad1.ca_address_sk AND
    ss_item_sk = i_item_sk AND ss_item_sk = sr_item_sk AND
    ss_ticket_number = sr_ticket_number AND ss_item_sk = cs_ui.cs_item_sk AND
    c_current_cdemo_sk = cd2.cd_demo_sk AND c_current_hdemo_sk = hd2.hd_demo_sk AND
    c_current_addr_sk = ad2.ca_address_sk AND c_first_sales_date_sk = d2.d_date_sk AND
    c_first_shipto_date_sk = d3.d_date_sk AND ss_promo_sk = p_promo_sk AND
    hd1.hd_income_band_sk = ib1.ib_income_band_sk AND
    hd2.hd_income_band_sk = ib2.ib_income_band_sk AND
    cd1.cd_marital_status <> cd2.cd_marital_status AND
    i_color IN ('purple', 'burlywood', 'indian', 'spring', 'floral', 'medium') AND
    i_current_price BETWEEN 64 AND 64 + 10 AND i_current_price BETWEEN 64 + 1 AND 64 + 15
  GROUP BY i_product_name, i_item_sk, s_store_name, s_zip, ad1.ca_street_number,
    ad1.ca_street_name, ad1.ca_city, ad1.ca_zip, ad2.ca_street_number,
    ad2.ca_street_name, ad2.ca_city, ad2.ca_zip, d1.d_year, d2.d_year, d3.d_year)
```

```sql
SELECT
  cs1.product_name,
  cs1.store_name,
  cs1.store_zip,
  cs1.b_street_number,
  cs1.b_streen_name,
  cs1.b_city,
  cs1.b_zip,
  cs1.c_street_number,
  cs1.c_street_name,
  cs1.c_city,
  cs1.c_zip,
  cs1.syear,
  cs1.cnt,
  cs1.s1,
  cs1.s2,
  cs1.s3,
  cs2.s1,
  cs2.s2,
  cs2.s3,
  cs2.syear,
  cs2.cnt
FROM cross_sales cs1, cross_sales cs2
WHERE cs1.item_sk = cs2.item_sk AND
  cs1.syear = 1999 AND
  cs2.syear = 1999 + 1 AND
  cs2.cnt <= cs1.cnt AND
  cs1.store_name = cs2.store_name AND
  cs1.store_zip = cs2.store_zip
ORDER BY cs1.product_name, cs1.store_name, cs2.cnt
```

Query Analysis – Q64 CBO ON



Query Analysis – Q64 CBO OFF



Overview

- Motivation
- Statistics Collection Framework
- Cost Based Optimizations
- TPC-DS Benchmark and Query Analysis
- Current Status and Future Work

Current Status

- SPARK-16026 is the umbrella jira.
  - A big project started from July 2016
  - 36 sub-tasks have been resolved
  - 50+ pull requests have been submitted
  - 10+ Spark contributors involved
- Good framework to allow integrations
  - Use statistics to derive if a join attribute is unique
  - Benefit star schema detection and its integration into join reorder

Try out CBO

- We encourage you to use CBO with Spark 2.2!
  - Configured via `spark.sql.cbo.enabled` (off by default)
- CBO has been available with Huawei FusionInsight HD since May 2016.
  - Our Spark CBO contribution is based on Huawei's CBO version.
- You can also try it on **Huawei Cloud**
  - UQuery (数据查询服务): free for now

Future Work

- Advanced statistics: e.g. histograms, sketches.
- Partition level statistics.
- Hint mechanism.
- Enhanced cost formula.

THANK YOU

- wangzhenhua@huawei.com