Making Apache Spark SQL Fast with Dynamic Partition Pruning

Bogdan Ghit

UvA, February 2020

databricks

2018-present Software Engineer at Databricks

- Performance optimizations in the SQL-engine
- Cloud infrastructure for Business Intelligence Workloads

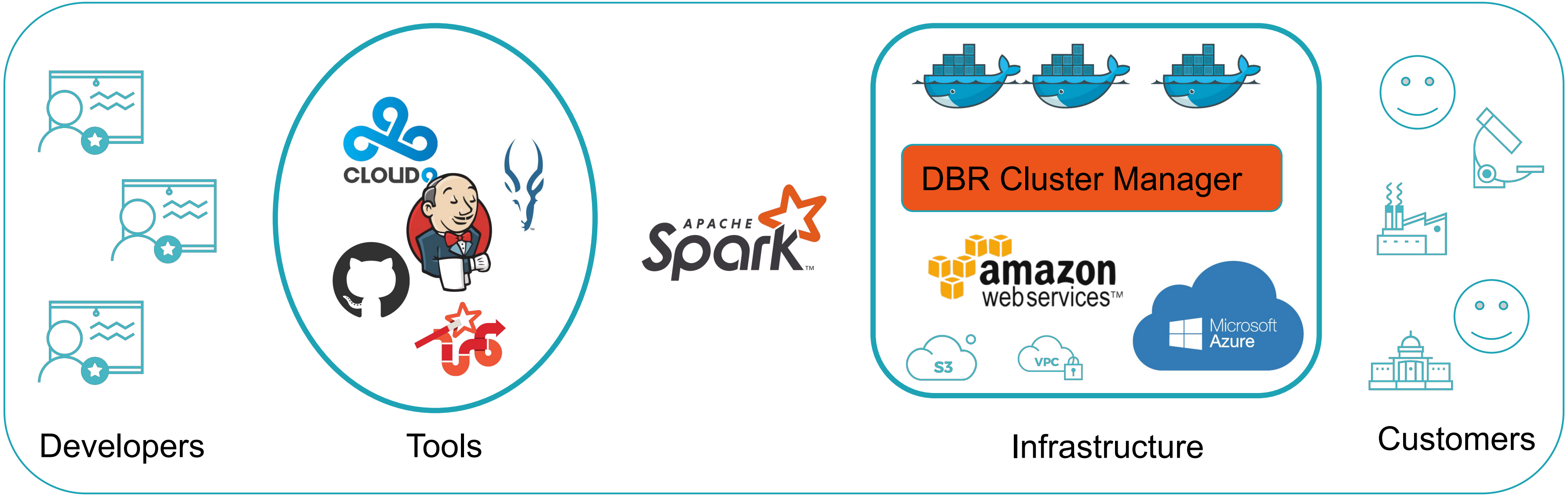
2012-2017 PhD in Computer Science from TU Delft

- Scheduling and resource allocation for big data frameworks
- Algorithmic aspects that arise in datacenters

2016 Research Intern at IBM Research T.J. Watson

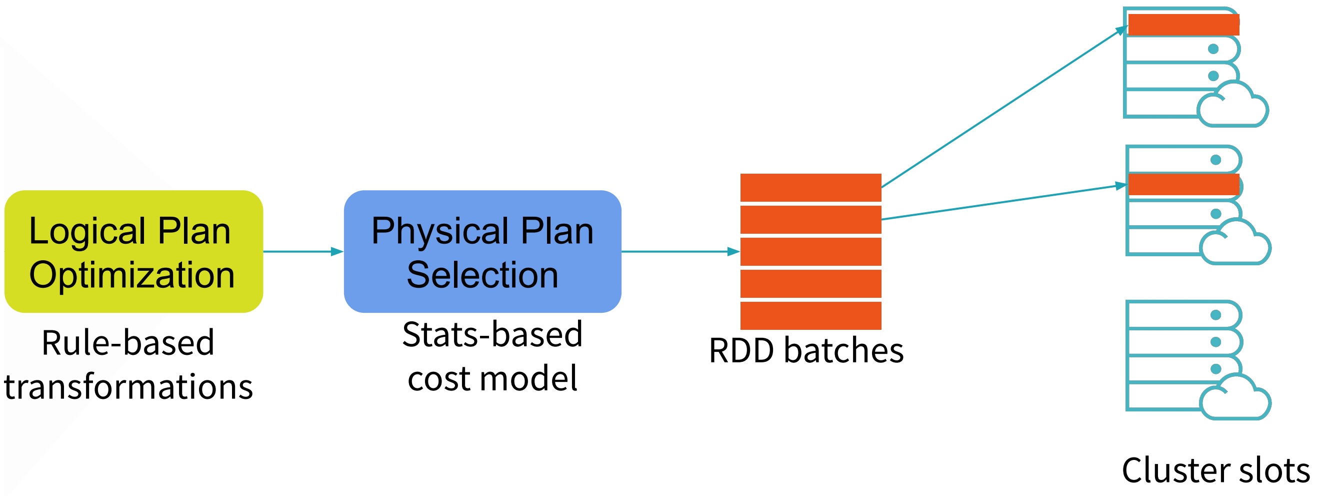
- Spot instance bid performance model
- Intersection of queueing theory and experimentation

Databricks Ecosystem



databricks

Spark In a Nutshell



Logical Plan Optimization

Rule-based transformations

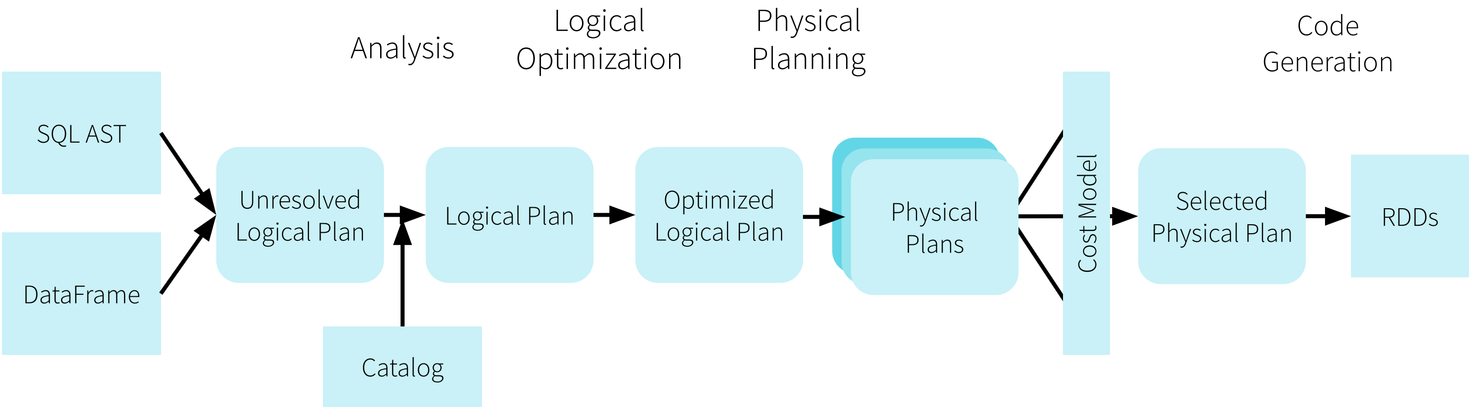
Physical Plan Selection

Stats-based cost model

RDD batches

Cluster slots

Catalyst as a Query Compiler



Catalyst is a functional, extensible query optimizer used by Spark SQL.

- Leverages advanced FP language (Scala) features
- Contains a library for representing trees and applying rules on them

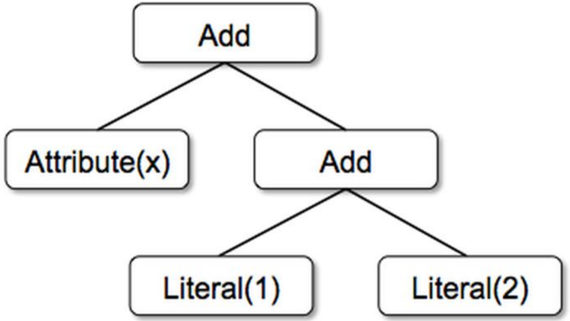
Trees in Catalyst

Tree is the main data structure used in Catalyst

- A tree is composed of node objects
- A node has a node type and zero or more children
- Node types are defined in Scala as subclasses of the `TreeNode` class

Examples:

- `Literal(value: Int)`
- `Attribute(name: String)`
- `Add(left: TreeNode, right: TreeNode)`



```
Add(Attribute(x), Add(Literal(1), Literal(2)))
```

Rules in Catalyst

Rules are functions that transform trees

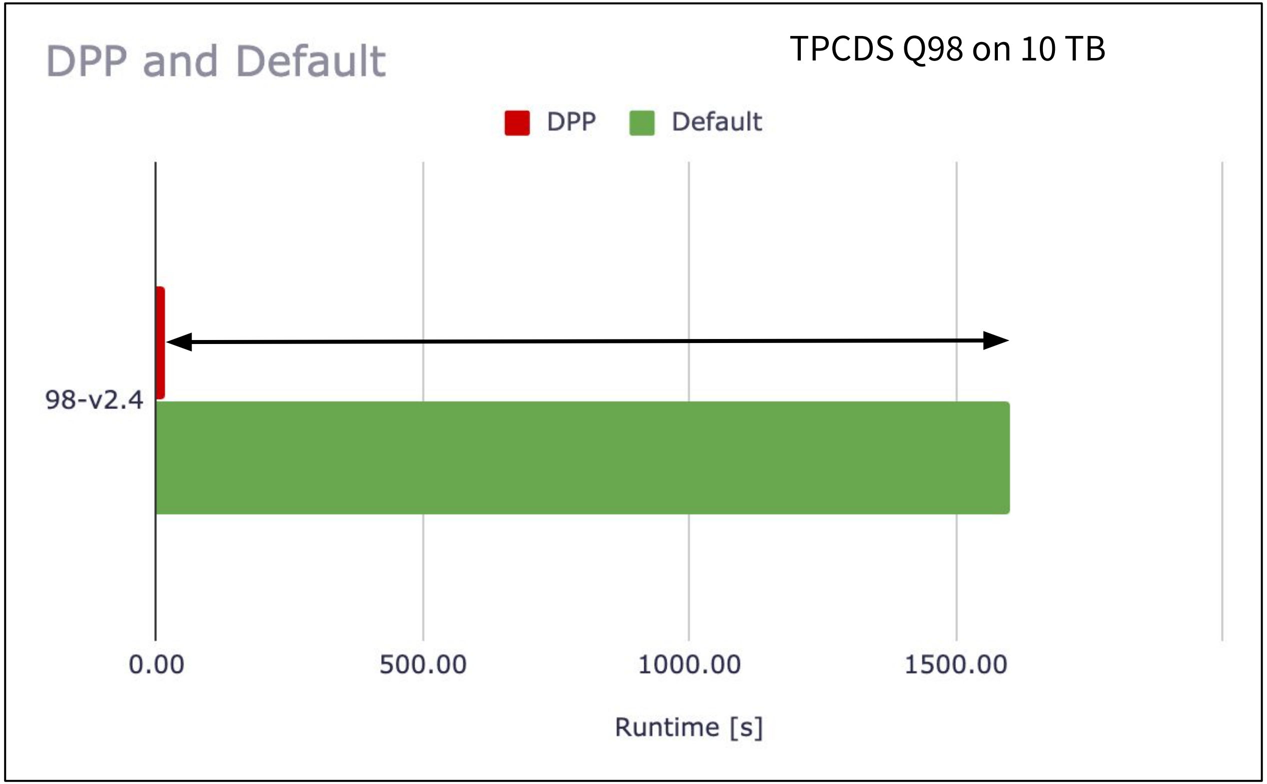
- Typically functional, leverage pattern matching
- `TreeNode.transformDown` (pre-order traversal)
- `TreeNode.transformUp` (post-order traversal)

TRANSFORMATION

```
tree.transform {
  case Add(Literal(c1), Literal(c2)) => Literal(c1 + c2)
  case Add(left, Literal(0)) => left
  case Add(Literal(0), right) => right
}
```

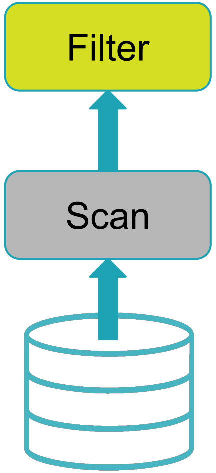
PATTERN

How to Make a Query 100x Faster?



Static Partition Pruning

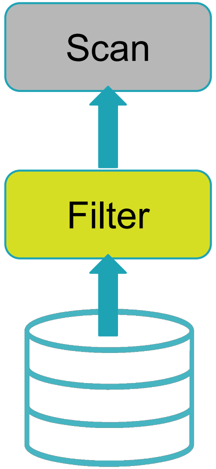
```
SELECT * FROM Sales WHERE day_of_week = ‘Mon’
```



Filter

Scan

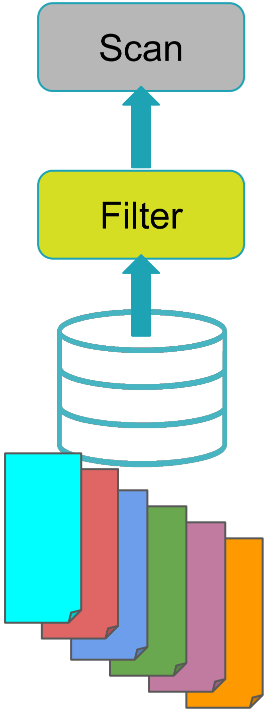
Basic data-flow



Scan

Filter

Filter Push-down



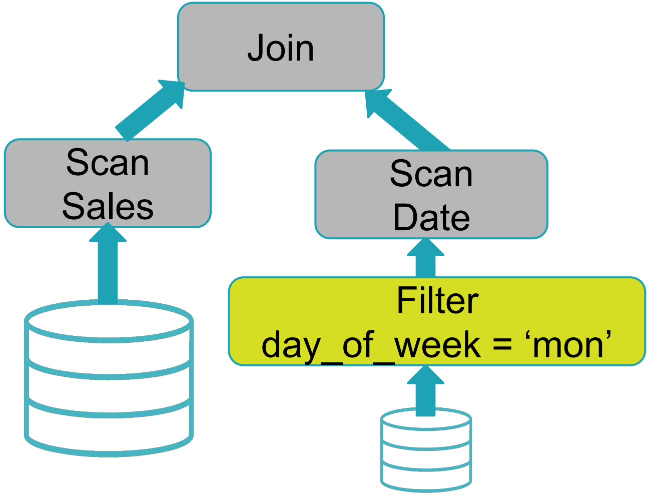
Scan

Filter

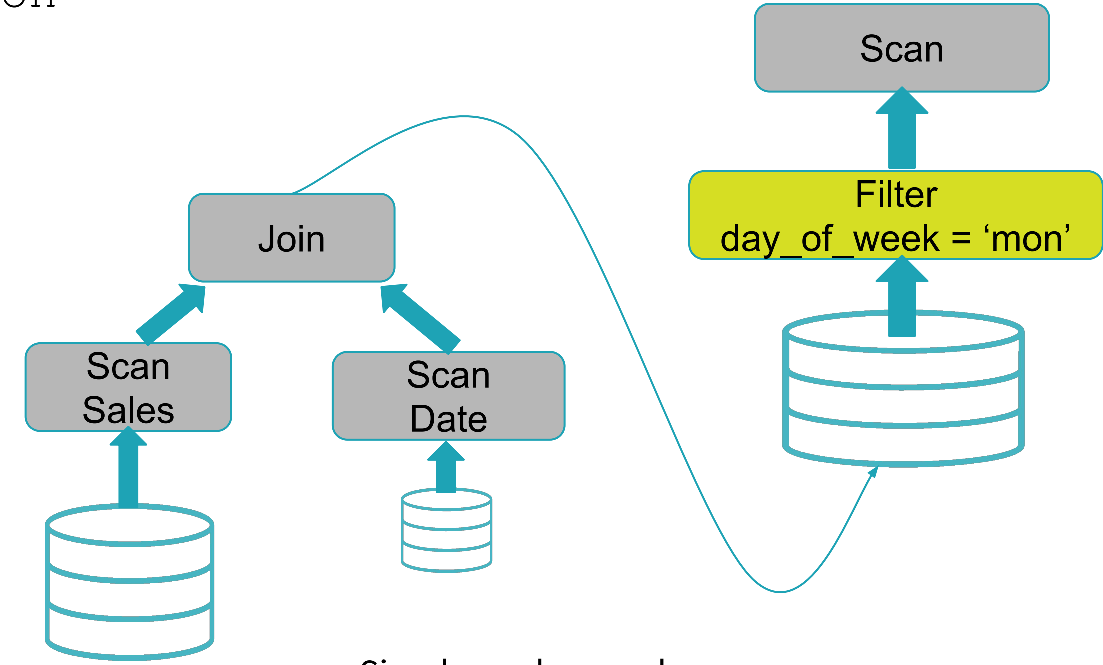
Partition files with multi-columnar data

Table Denormalization

```
SELECT * FROM Sales JOIN Date
WHERE Date.day_of_week = ‘Mon’
```



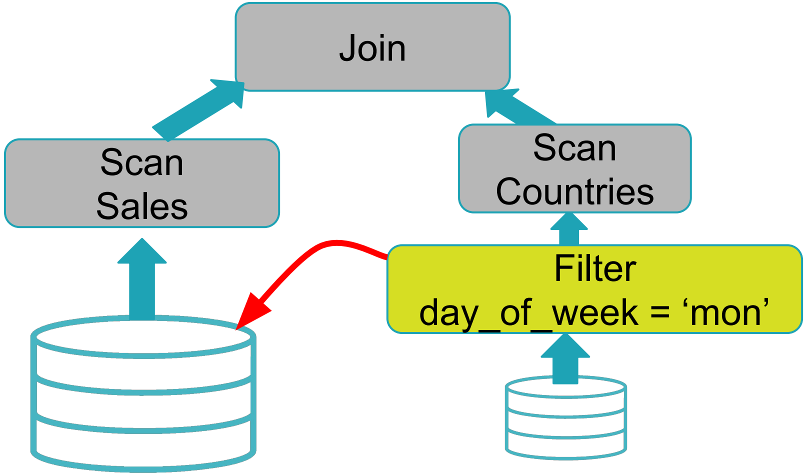
Static pruning not possible



Simple workaround

This Talk

```
SELECT * FROM Sales JOIN Date
WHERE Date.day_of_week = 'Mon'
```

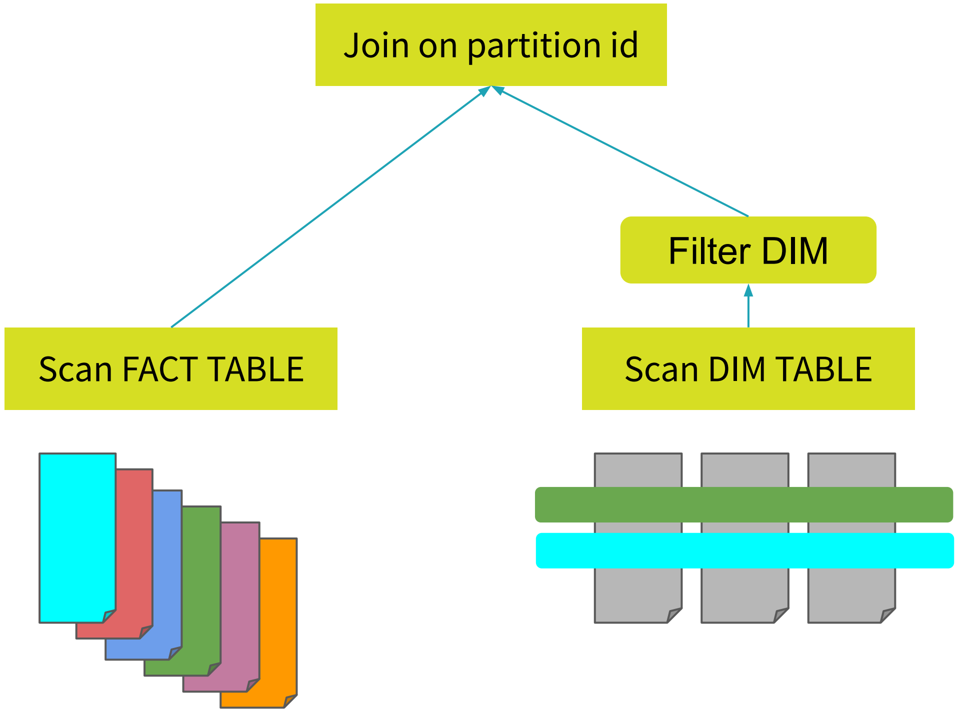


Dynamic pruning

Optimization Opportunities

Query Shape

Data Layout



Join on partition id

Filter DIM

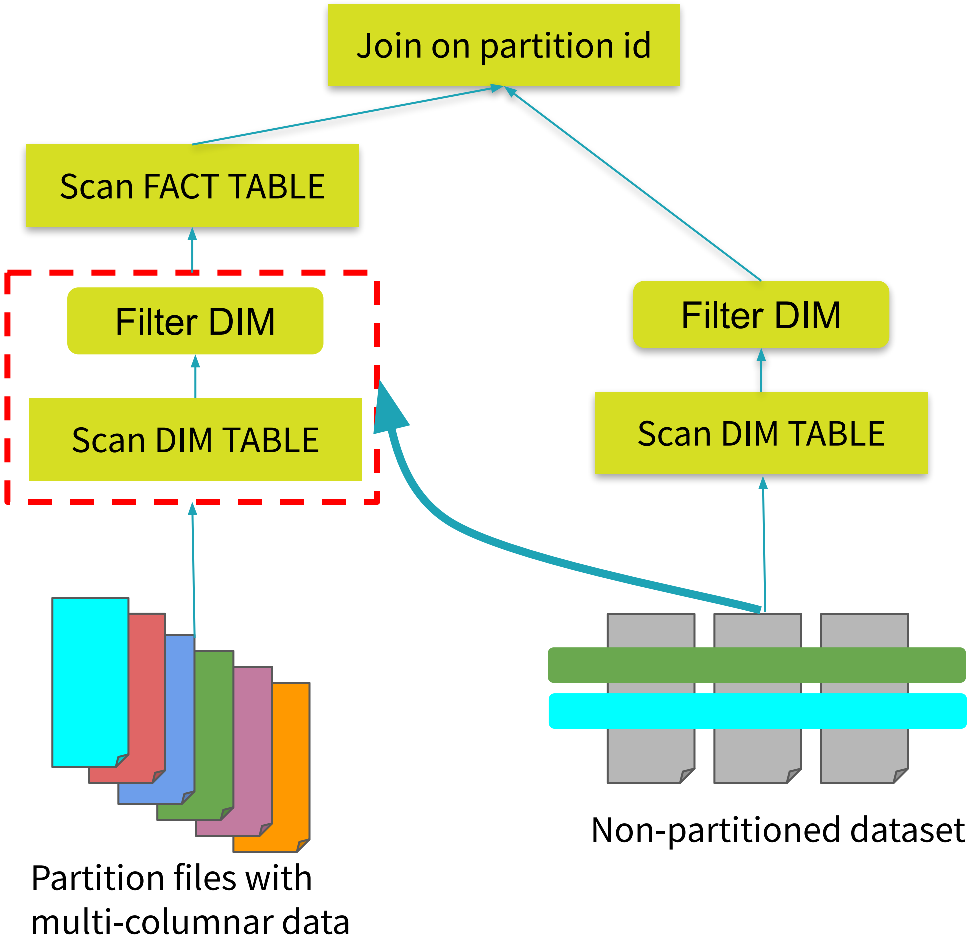
Scan FACT TABLE

Scan DIM TABLE

Partition files with
multi-columnar data

Non-partitioned dataset

A Simple Approach



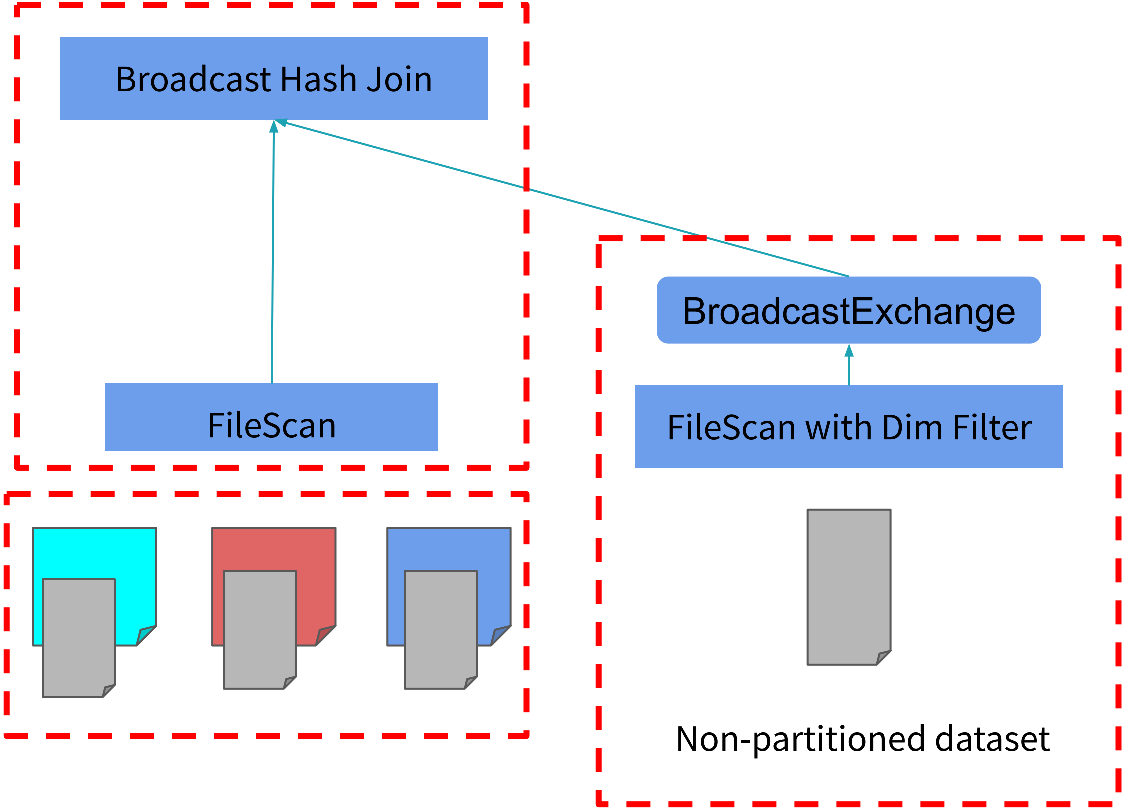
Work duplication may be expensive

Heuristics based on inaccurate stats

Broadcast Hash Join

Execute the join
locally without
a shuffle

Broadcast the build
side results



Broadcast Hash Join

FileScan

BroadcastExchange

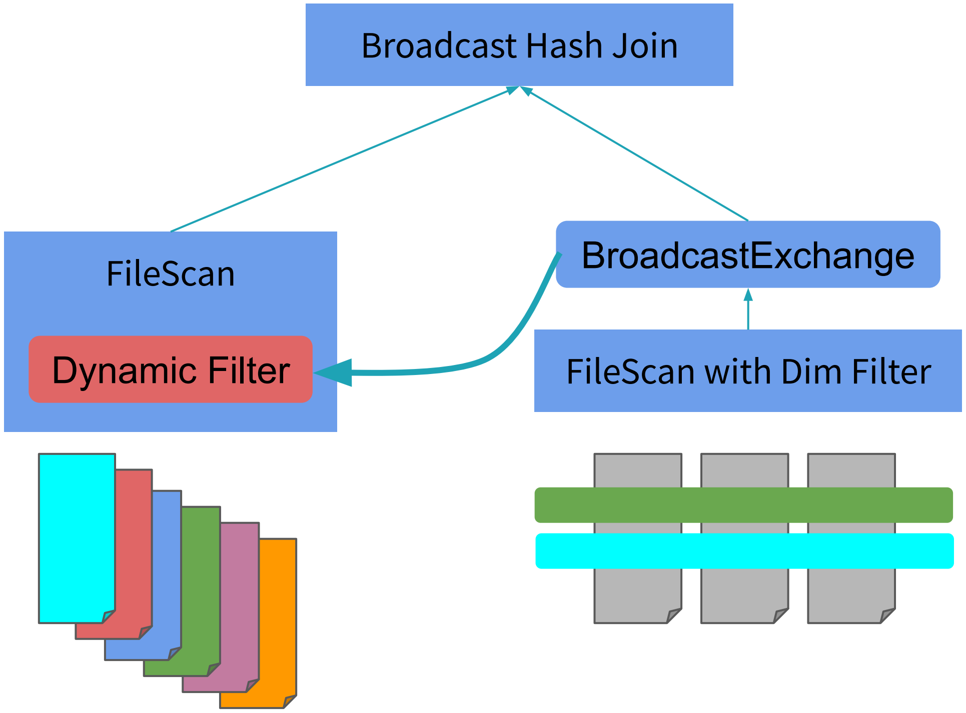
FileScan with Dim Filter

Non-partitioned dataset

Execute the build side
of the join

Place the result in a
broadcast variable

Reusing Broadcast Results



Broadcast Hash Join

FileScan

Dynamic Filter

BroadcastExchange

FileScan with Dim Filter

Partition files with multi-columnar data

Non-partitioned dataset

Experimental Setup

Workload Selection

- TPC-DS scale factors 1-10 TB

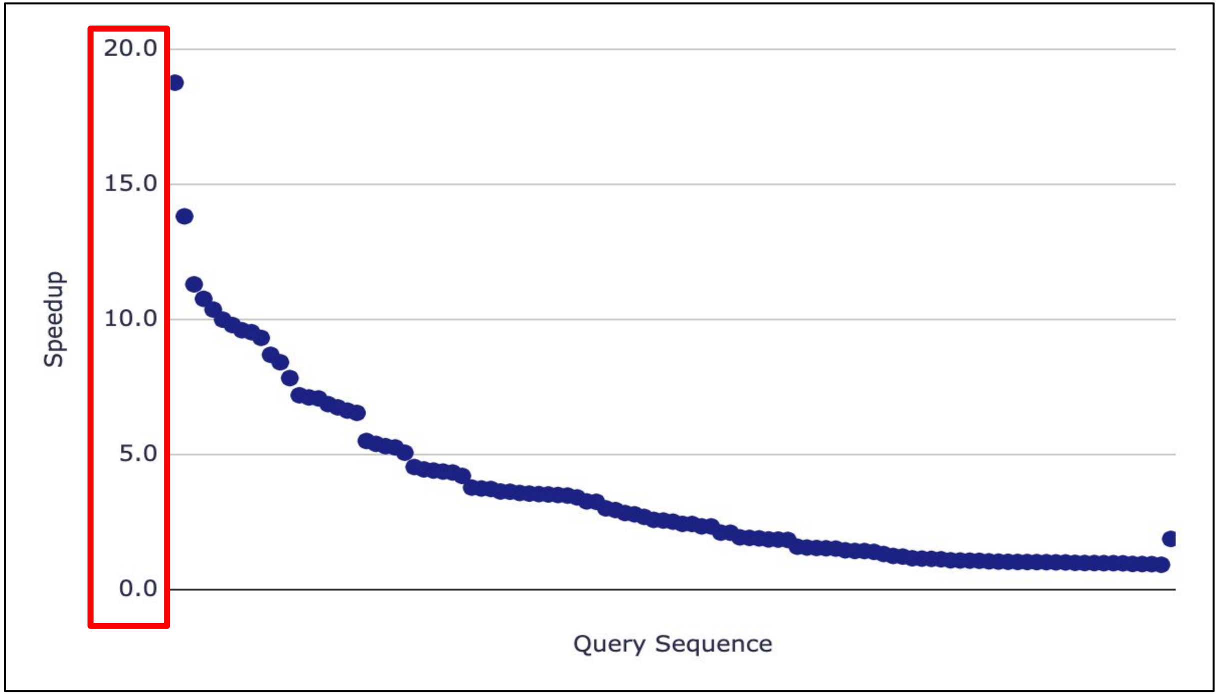
Cluster Configuration

- 10 i3.xlarge machines

Data-Processing Framework

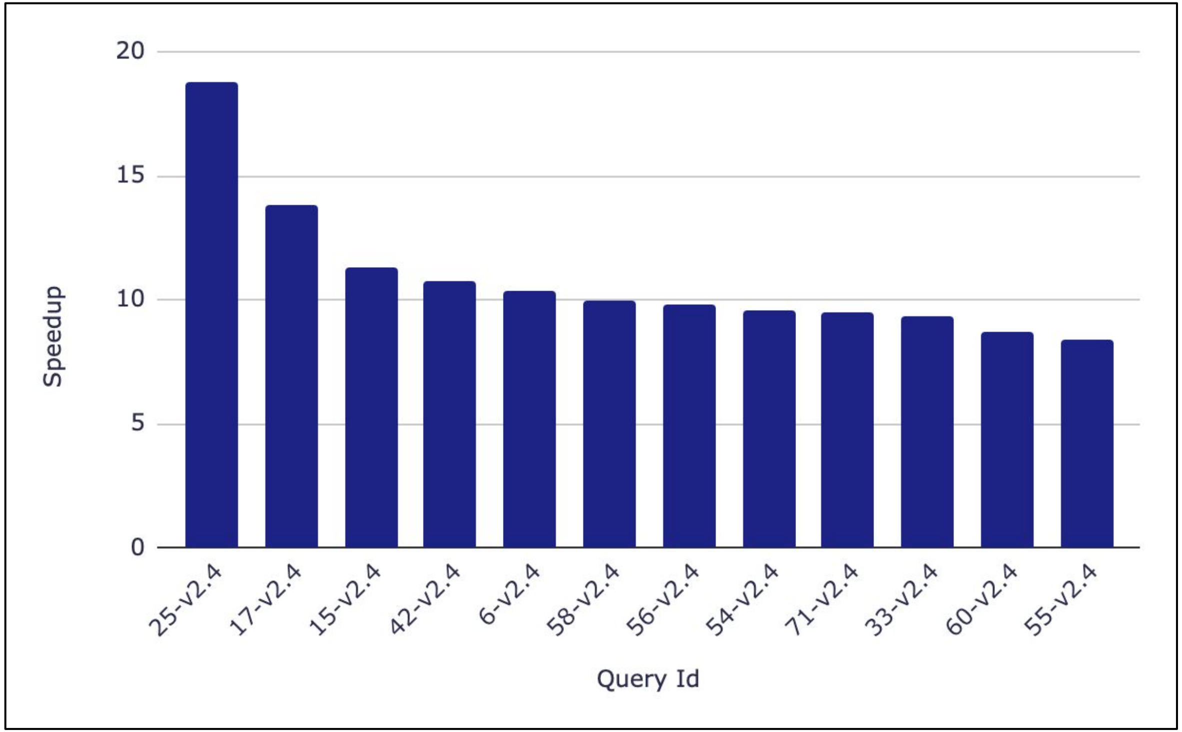
- Apache Spark 3.0

TPCDS 1 TB



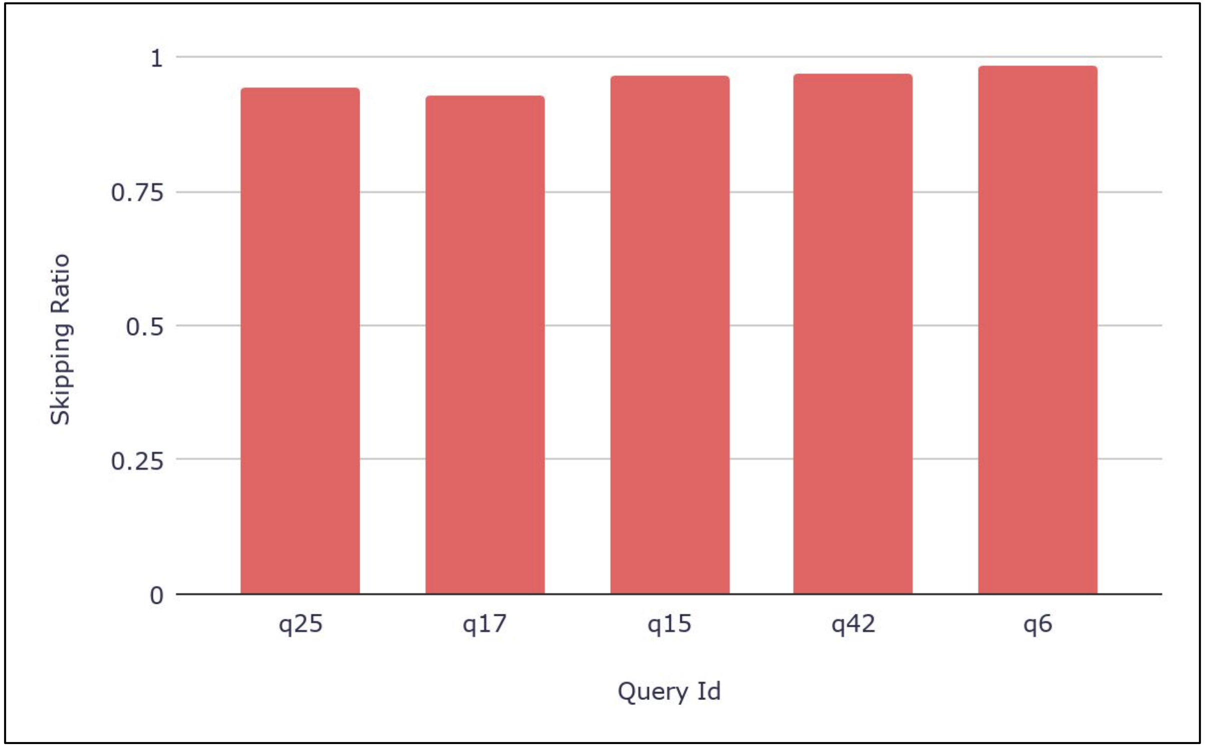
60 / 102 queries speedup between 2 and 18

Top Queries



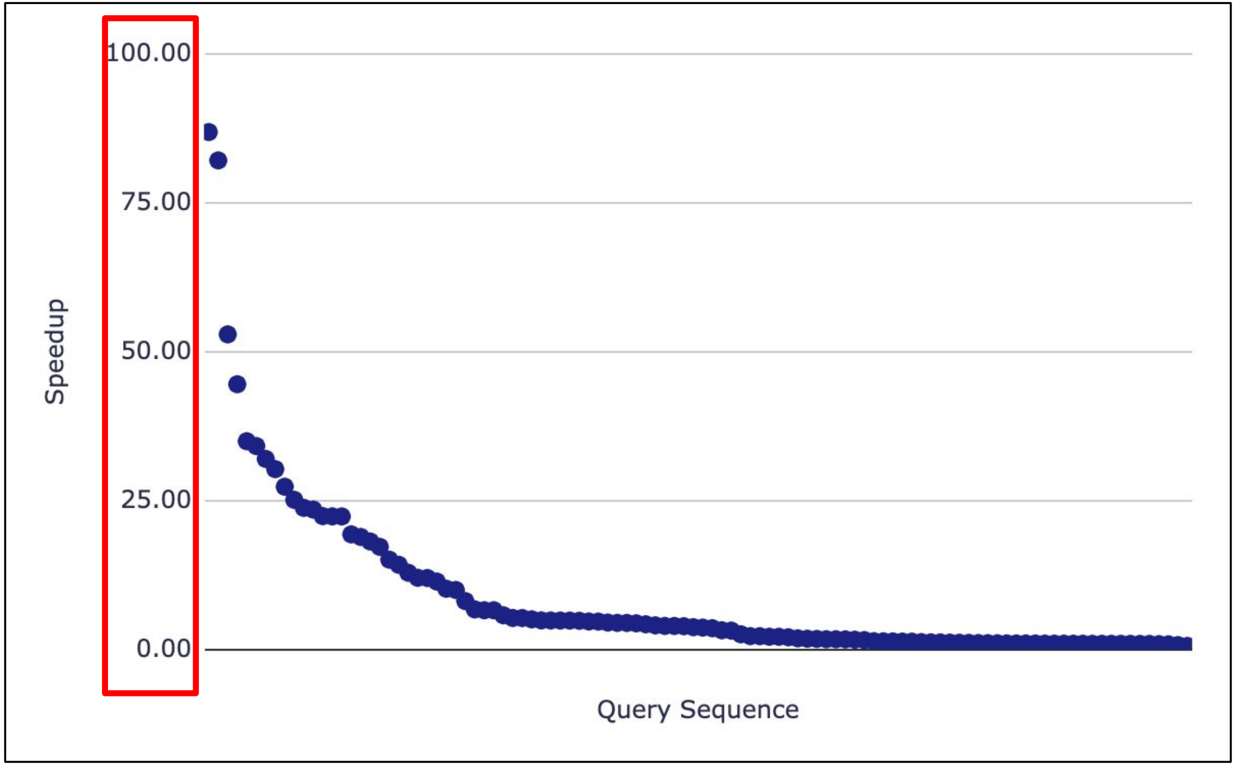
Very good speedups for top 10% of the queries

Data Skipped



Very effective in skipping data

TPCDS 10 TB

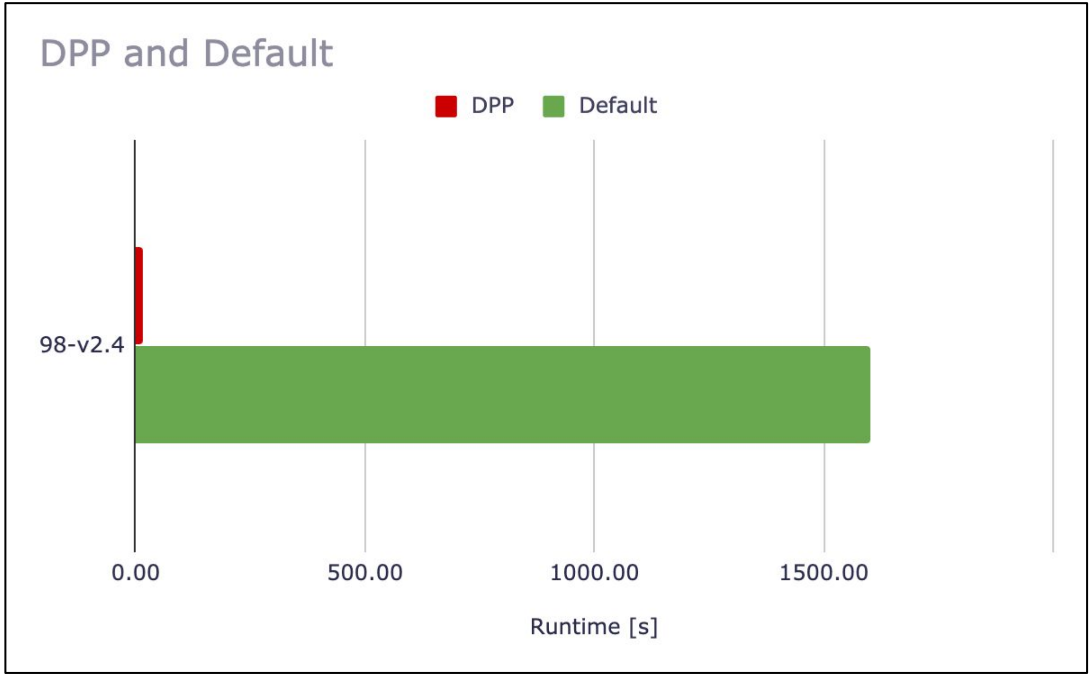


Even better speedups at 10x the scale

Query 98

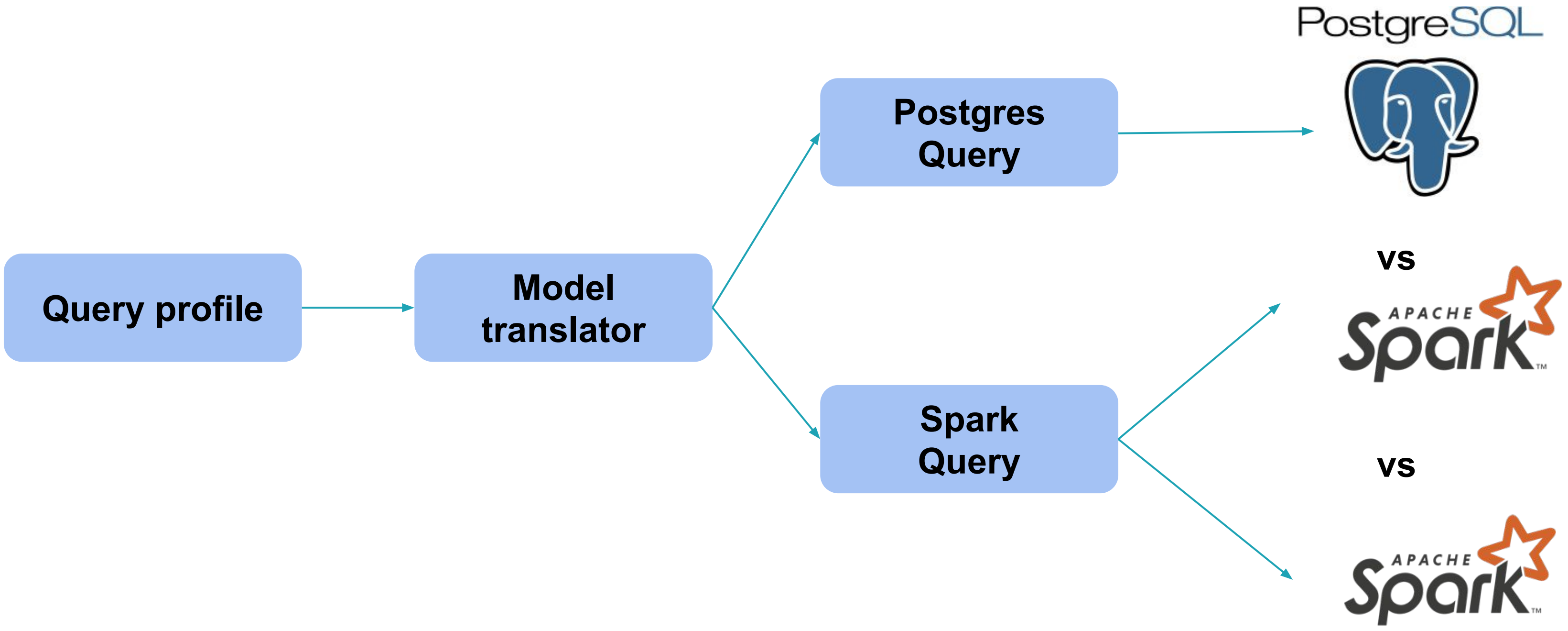
```
SELECT i_item_desc, i_category, i_class, i_current_price,
       sum(ss_ext_sales_price) as itemrevenue,
       sum(ss_ext_sales_price)*100/sum(sum(ss_ext_sales_price)) over
         (partition by i_class) as revenueratio
FROM
   store_sales, item, date_dim
WHERE
  ss_item_sk = i_item_sk
  and i_category in ('Sports', 'Books', 'Home')
  and ss_sold_date_sk = d_date_sk
  and cast(d_date as date) between cast('1999-02-22' as date)
           and (cast('1999-02-22' as date) + interval '30' day)
GROUP BY
  i_item_id, i_item_desc, i_category, i_class, i_current_price

ORDER BY
  i_category, i_class, i_item_id, i_item_desc, revenueratio
```

TPCDS 10 TB

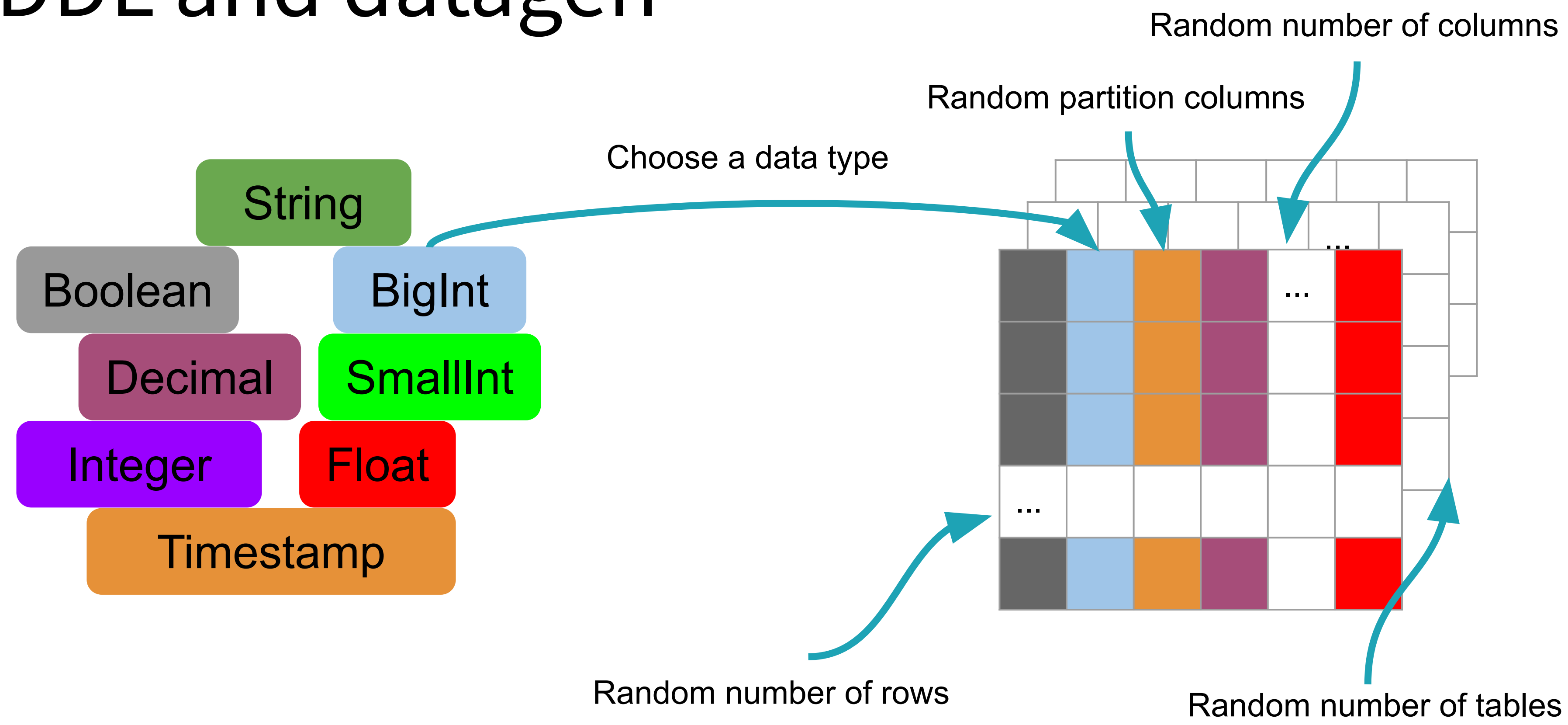


Highly selective dimension filter that retains only one month out of 5 years of data

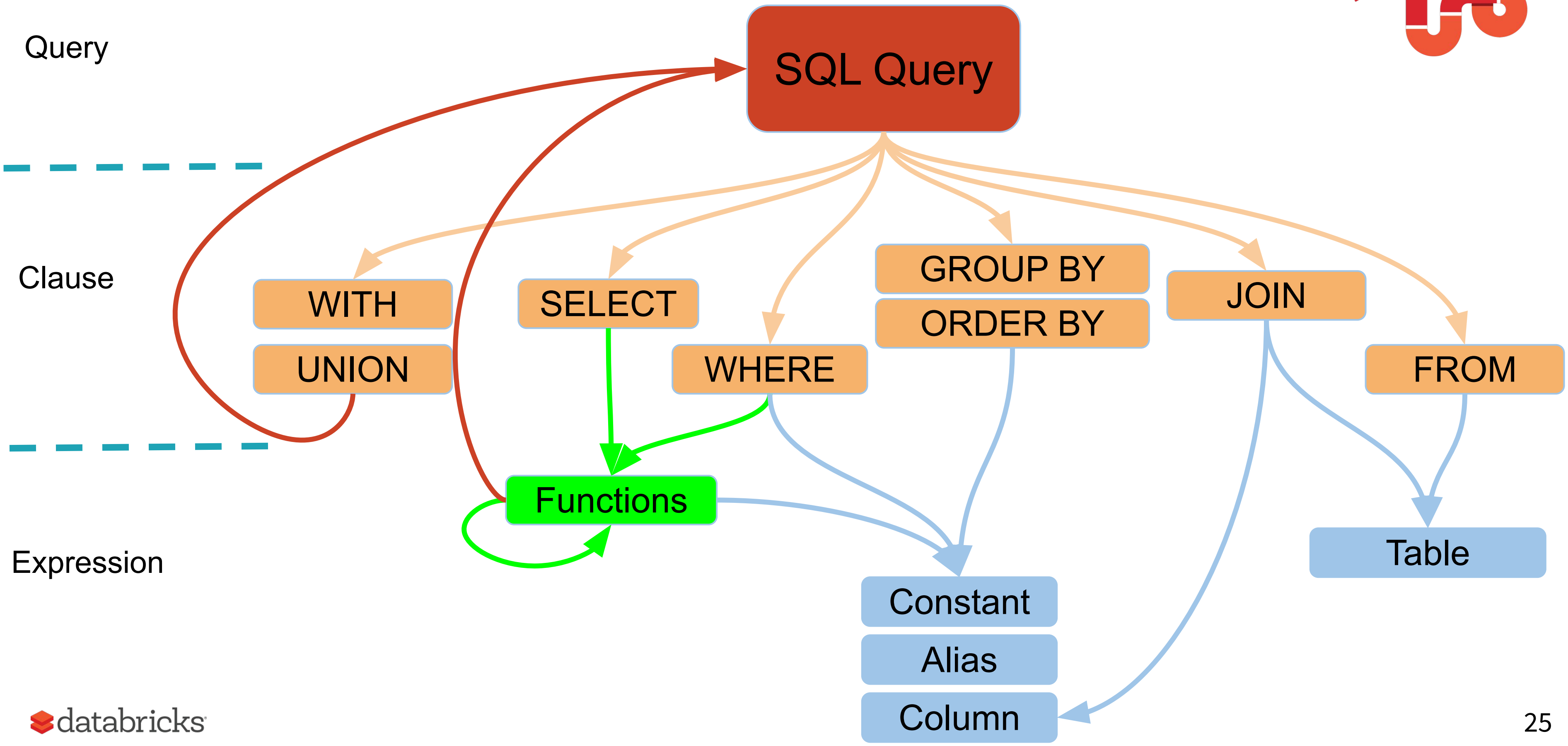
Random query generation



DDL and datagen



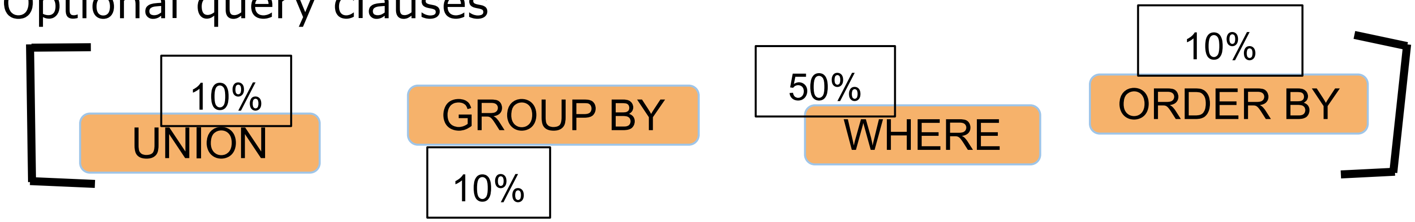
Recursive query model



Probabilistic query profile

Independent weights

- Optional query clauses



10%
UNION

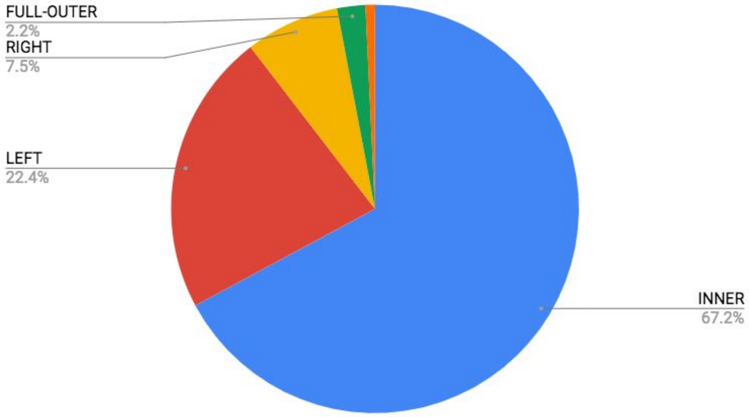
GROUP BY
10%

50%
WHERE

10%
ORDER BY

Inter-dependent weights

- Join types
- Select functions



databricks

Coalesce flattening (1/4)

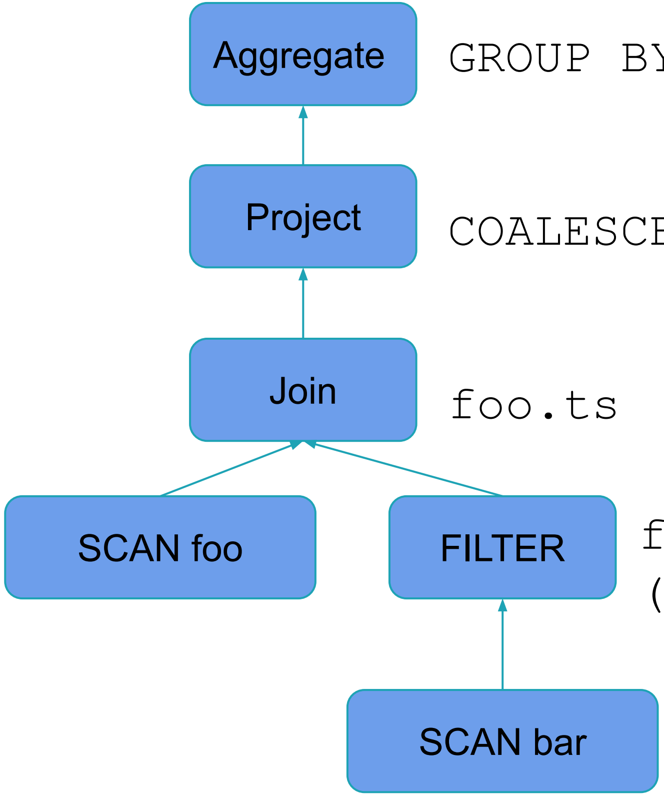
```sql
SELECT COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3) AS int_col,
       IF(NULL, VARIANCE(COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)),
       COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)) AS int_col_1,
       STDDEV(t2.double_col_2) AS float_col,
       COALESCE(MIN((t1.smallint_col_3) - (COALESCE(t2.smallint_col_3, t1.smallint_col_3,
       t2.smallint_col_3))), COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3),
       COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)) AS int_col_2
FROM table_4 t1
INNER JOIN table_4 t2 ON (t2.timestamp_col_7) = (t1.timestamp_col_7)
WHERE (t1.smallint_col_3) IN (CAST('0.04' AS DECIMAL(10,10)), t1.smallint_col_3)
GROUP BY COALESCE(t2.smallint_col_3, t1.smallint_col_3, t2.smallint_col_3)
```

Small dataset with 2 tables of 5x5 size

Within 10 randomly generated queries

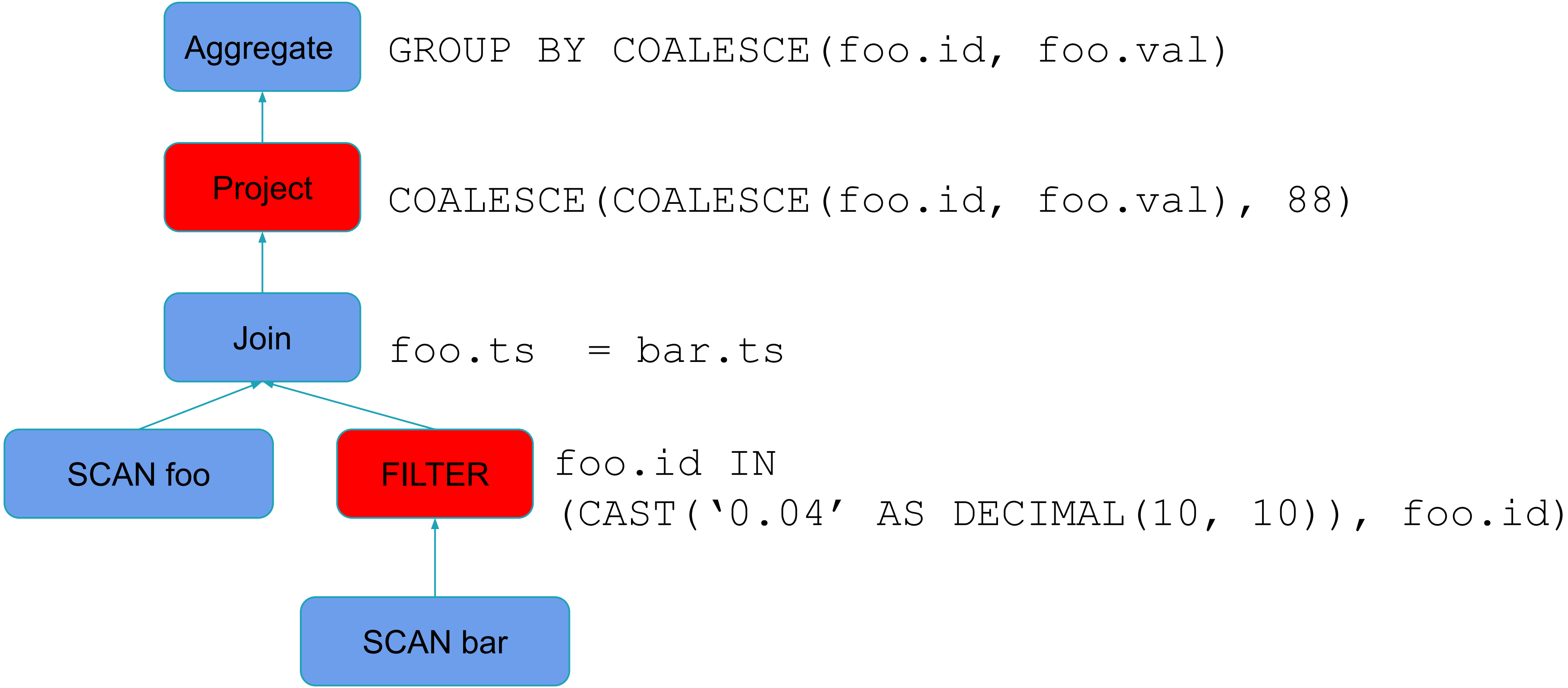
Error: Operation is in ERROR_STATE

Coalesce flattening (2/4)



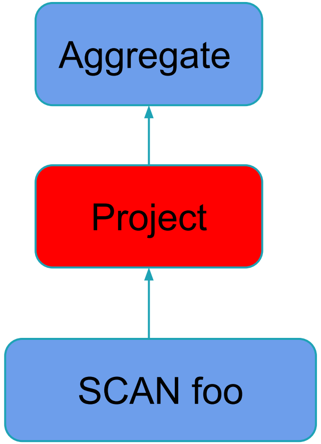
```
GROUP BY COALESCE(foo.id, foo.val)

COALESCE(COALESCE(foo.id, foo.val), 88)

foo.ts  = bar.ts

foo.id IN
(CAST(‘0.04’ AS DECIMAL(10, 10)), foo.id)
```

Coalesce flattening (3/4)



```
GROUP BY COALESCE(foo.id, foo.val)

COALESCE(COALESCE(foo.id, foo.val), 88)

foo.ts  = bar.ts

foo.id IN
(CAST(‘0.04’ AS DECIMAL(10, 10)), foo.id)
```

Coalesce flattening (4/4)



Minimized query:

```
SELECT
   COALESCE(COALESCE(foo.id, foo.val), 88)
FROM foo
GROUP BY
   COALESCE(foo.id, foo.val)
```

Analyzing the error

- The optimizer flattens the nested coalesce calls
- The SELECT clause doesn't contain the GROUP BY expression
- Possibly a problem with any GROUP BY expression that can be optimized

Conclusion

Apache Spark 3.0 introduces Dynamic Partition Pruning

- Strawman approach at logical planning time
- Optimized approach during execution time

Significant speedup, exhibited in many TPC-DS queries

With this optimization Spark may now work good with star-schema queries, making it unnecessary to ETL denormalized tables.

Bogdan Ghit - https://bogdanghit.github.io/