



Deep Dive Into APACHE *Spark*™ *SQL*

with Advanced Performance Tuning

Xiao Li & Wenchen Fan

Spark Summit | SF | Jun 2018



About US

- Software Engineers at  databricks
- Apache Spark Committers and PMC Members



Xiao Li (Github: [gatorsmile](#))



Wenchen Fan (Github: [cloud-fan](#))

Databricks' Unified Analytics Platform

Unifies Data Engineers
and Data Scientists

COLLABORATIVE NOTEBOOKS




Data Engineers



Data Scientists

Unifies Data and AI
Technologies

DATABRICKS RUNTIME

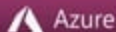
Powered by 

Delta SQL Streaming PYTORCH  Studio  TensorFlow 

Eliminates infrastructure
complexity



CLOUD NATIVE SERVICE



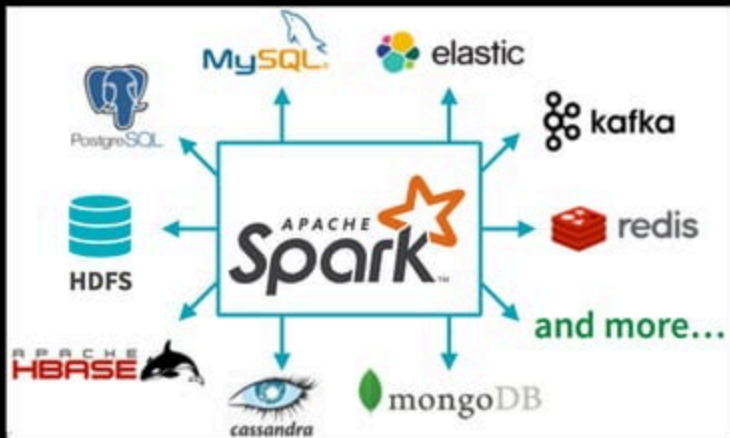
Spark SQL

A **highly scalable** and **efficient** relational processing engine with **ease-to-use APIs** and **mid-query fault tolerance**.



Run Everywhere

Processes, integrates and analyzes the data from diverse data sources (e.g., Cassandra, Kafka and Oracle) and file formats (e.g., Parquet, ORC, CSV, and JSON)



The not-so-secret truth...

 **Spark** SQL is **not only** SQL.



Structured Streaming

MLlib

GraphFrame

TensorFrames

SQL

SparkSession / DataFrame / Dataset APIs

Catalyst Optimization & Tungsten Execution

Data Source Connectors

Spark Core (RDD APIs)

Spark SQL



Not Only SQL

Powers and **optimizes** the other Spark applications and libraries:

- **Structured streaming** for stream processing
- **MLlib** for machine learning
- **GraphFrame** for graph-parallel computation
- **Your own Spark applications** that use SQL, DataFrame and Dataset APIs

Lazy Evaluation

Optimization happens **as late as possible**, therefore Spark SQL can optimize *across* functions and libraries

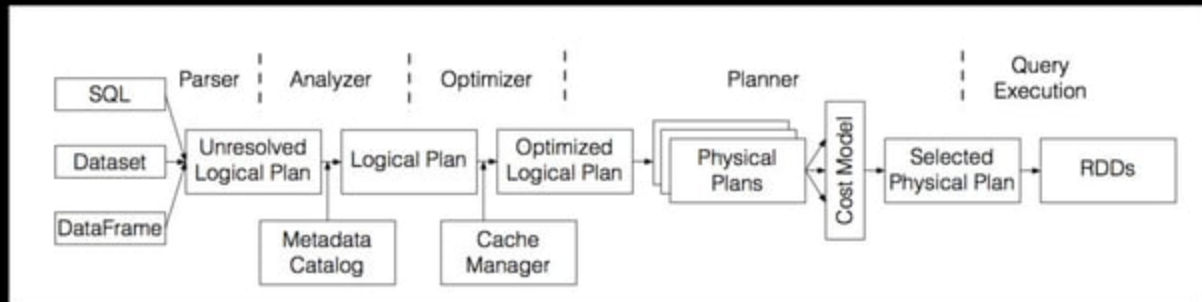
Holistic optimization when using these libraries and SQL/DataFrame/Dataset APIs in the same Spark application.

New Features of Spark SQL in Spark 2.3

- PySpark Pandas UDFs [[SPARK-22216](#)] [[SPARK-21187](#)]
- Stable Codegen [[SPARK-22510](#)] [[SPARK-22692](#)]
- Advanced pushdown for partition pruning predicates [[SPARK-20331](#)]
- Vectorized ORC reader [[SPARK-20682](#)] [[SPARK-16060](#)]
- Vectorized cache reader [[SPARK-20822](#)]
- Histogram support in cost-based optimizer [[SPARK-21975](#)]
- Better Hive compatibility [[SPARK-20236](#)] [[SPARK-17729](#)] [[SPARK-4131](#)]
- More efficient and extensible data source API V2

Spark SQL

A **compiler** from **queries** to **RDDs**.



Performance Tuning for Optimal Plans

Run EXPLAIN **Plan.**

Interpret **Plan.**

Tune **Plan.**

Get the plans by running Explain command/APIs, or the SQL tab in either Spark UI or Spark History Server



The screenshot shows the Spark UI interface with the 'SQL' tab selected. The main heading is 'Details for Query 111'. Below this, the following information is displayed: 'Submitted Time: 2018/05/27 18:14:46', 'Duration: 0.2 s', and 'Succeeded Jobs: 198 199'. There is a 'Details' link with a downward arrow icon.

```
== Physical Plan ==
*(3) HashAggregate(keys=[], functions=[finalmerge_count(merge count#2695L) AS count(1)#2691L], output=[count#2692L])
+- Exchange SinglePartition
  +- *(2) HashAggregate(keys=[], functions=[partial_count(1) AS count#2695L], output=[count#2695L])
    +- *(2) Project
      +- *(2) BroadcastNestedLoopJoin BuildRight, LeftSemi, (a1#2679L < b1#2683L)
        :- *(2) Project [id#2677L AS a1#2679L]
          : +- *(2) Range (0, 20000, step=1, splits=8)
        +- BroadcastExchange IdentityBroadcastMode
          +- *(1) Project [id#2681L AS b1#2683L]
            +- *(1) Range (0, 10000, step=1, splits=8)
```

More statistics from the Job page

Details for Job 198

Status: SUCCEEDED

Job Group: 864cb5a9-115e-4d69-99a

Completed Stages: 1

▶ Event Timeline

▼ DAG Visualization

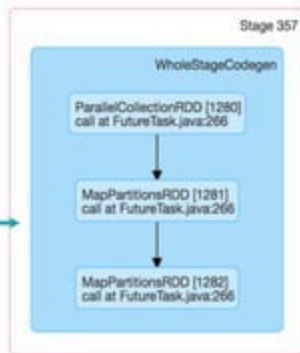


Details for Stage 357 (Attempt 0)

Total Time Across All Tasks: 5 ms

Locality Level Summary: Process local: 8

▼ DAG Visualization



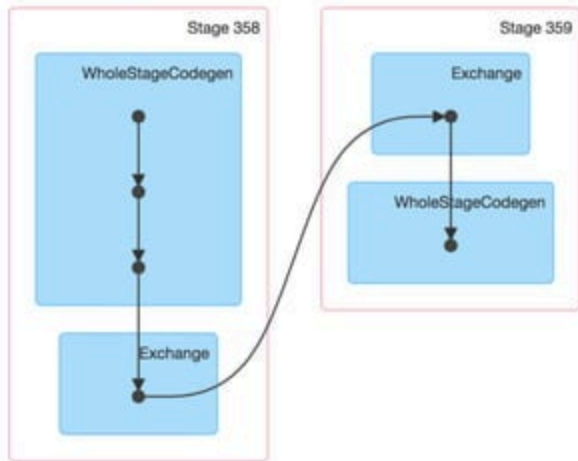
Details for Job 199

Status: SUCCEEDED

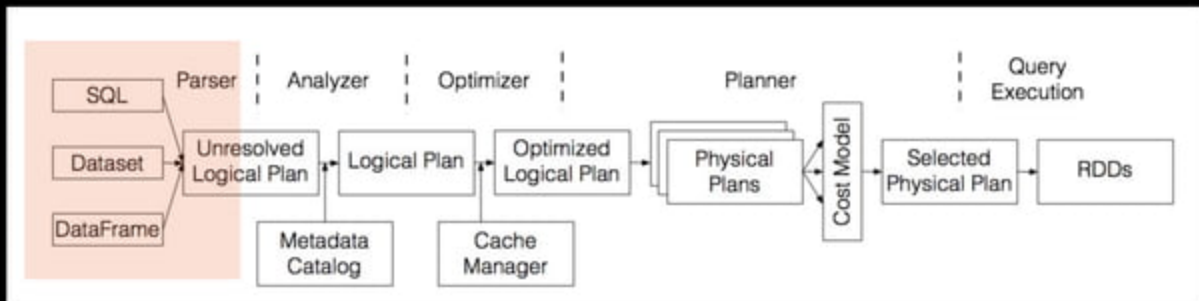
Completed Stages: 2

▶ Event Timeline

▼ DAG Visualization



Declarative APIs



Declarative APIs

Declare your intentions by

- **SQL** API: ANSI SQL:2003 and HiveQL.
- **Dataset/DataFrame** APIs: **richer, language-integrated** and **user-friendly** interfaces

Declarative APIs

When should I use SQL, DataFrames or Datasets?

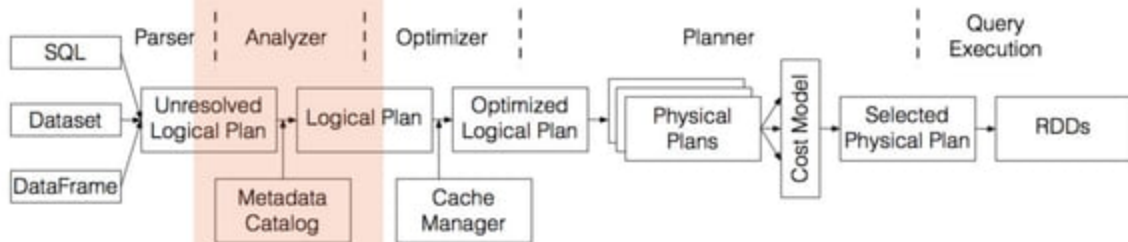
- The DataFrame API provides **untyped** relational operations
- The Dataset API provides a **typed** version, at the cost of performance due to heavy reliance on user-defined closures/lambda's.

[[SPARK-14083](#)]

- <http://dbricks.co/29xYnqR>

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Metadata Catalog



Metadata Catalog

- Persistent **Hive metastore** [Hive 0.12 - Hive 2.3.3]
- Session-local **temporary view manager**
- Cross-session **global temporary view manager**
- Session-local **function registry**

Metadata Catalog

Session-local **function registry**

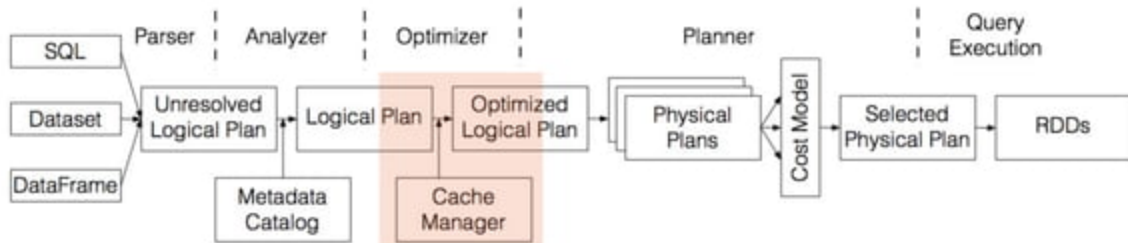
- Easy-to-use lambda UDF
- Vectorized PySpark Pandas UDF
- Native UDAF interface
- Support Hive UDF, UDAF and UDTF
- **Almost 300** built-in SQL functions
- Next, [SPARK-23899](#) adds 30+ **high-order built-in** functions.
- Blog for high-order functions: <https://dbricks.co/2rR8vAr>

Performance Tips - Catalog

Time costs of partition metadata retrieval:

- Upgrade your Hive metastore
- Avoid very high cardinality of partition columns
- Partition pruning predicates (improved in [[SPARK-20331](#)])

Cache Manager



Cache Manager

- Automatically replace by cached data when plan matching
- Cross-session
- Dropping/Inserting tables/views invalidates all the caches that depend on it
- Lazy evaluation

Performance Tips

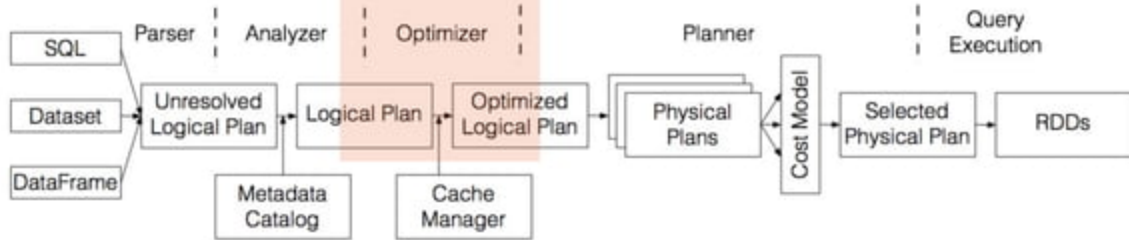
Cache: not always fast if spilled to disk.

- Uncache it, if not needed.

Next releases:

- A new cache mechanism for building the **snapshot** in cache. Querying stale data. Resolved by names instead of by plans. [[SPARK-24461](#)]

Optimizer



Optimizer

Rewrites the query plans using **heuristics** and **cost**.

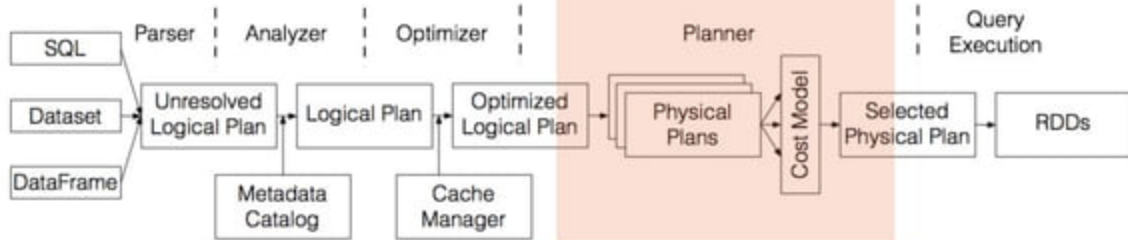
- Column pruning
 - Predicate push down
 - Constant folding
 - Outer join elimination
 - Constraint propagation
 - Join reordering
- and many more.

Performance Tips

Roll your own Optimizer and Planner Rules

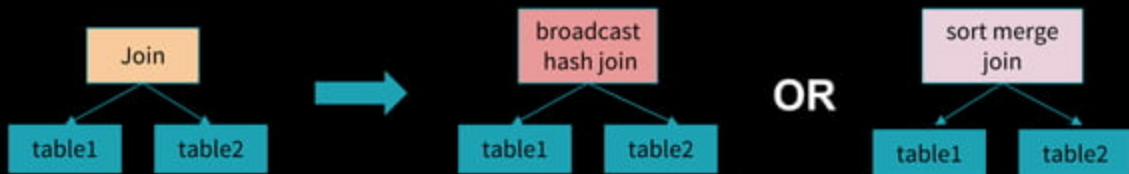
- In class ExperimentalMethods
 - **var** *extraOptimizations*: Seq[Rule[LogicalPlan]] = Nil
 - **var** *extraStrategies*: Seq[Strategy] = Nil
- Examples in the Herman's talk *Deep Dive into Catalyst Optimizer*
 - Join two intervals: <http://dbricks.co/2etjIDY>

Planner



Planner

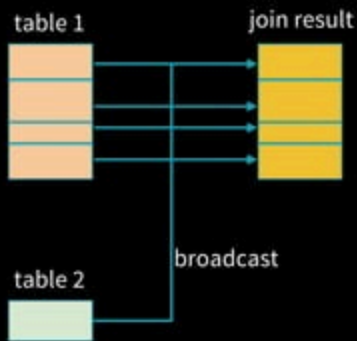
- Turn logical plans to physical plans. (what to how)
- Pick the best physical plan according to the cost



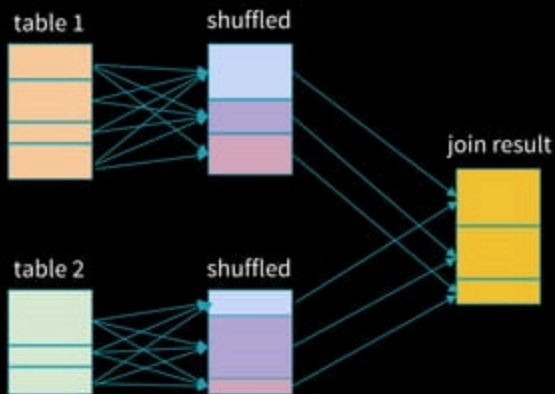
broadcast join has lower cost if
one table can fit in memory

Performance Tips - Join Selection

broadcast join



shuffle join



Performance Tips - Join Selection

broadcast join vs **shuffle join** (broadcast is faster)

- `spark.sql.autoBroadcastJoinThreshold`
- Keep the statistics updated
- `broadcastJoin` Hint

Performance Tips - Equal Join

```
... t1 JOIN t2 ON t1.id = t2.id AND t1.value < t2.value
```

```
... t1 JOIN t2 ON t1.value < t2.value
```

Put at least one equal predicate in join condition

Performance Tips - Equal Join

... t1 JOIN t2 ON t1.id = t2.id AND t1.value < t2.value

... t1 JOIN t2 ON t1.value < t2.value

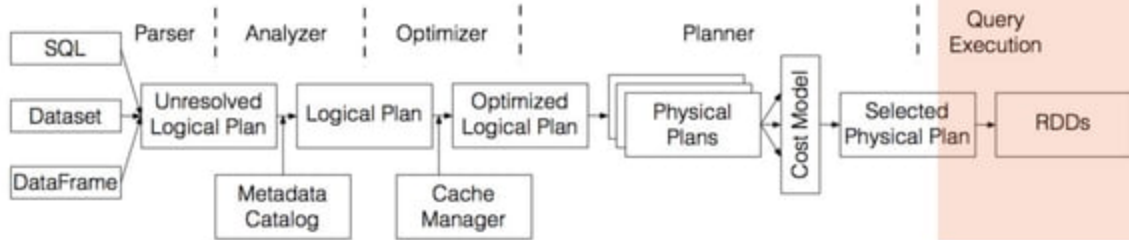
```
for l in left:
    for r in right:
        if (satisfy_join_condition(l, r)):
            yield join(l, r)
```

$O(n^2)$

```
hash_relation = build_relation(l)
for r in right:
    if (hash_relation.contains(r.joinKey)):
        l = hash_relation.get(r.joinKey)
        if (satisfy_join_condition(l, r)):
            yield join(l, r)
```

$O(n)$

Query Execution



Query Execution

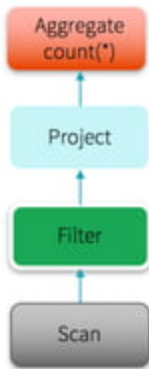
- **Memory Manager**: tracks the memory usage, efficiently distribute memory between tasks/operators.
- **Code Generator**: compiles the physical plan to optimal java code.
- **Tungsten Engine**: efficient binary data format and data structure for CPU and memory efficiency.

Performance Tips - Memory Manager

Tune `spark.executor.memory` and `spark.memory.fraction` to leave enough space for unsupervised memory. Some memory usages are NOT tracked by Spark (netty buffer, parquet writer buffer).

Set `spark.memory.offHeap.enabled` and `spark.memory.offHeap.size` to enable offheap, and decrease `spark.executor.memory` accordingly.

Whole Stage Code Generation



```
long count = 0;
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1;
  }
}
```

Performance Tip - WholeStage codegen

Tune `spark.sql.codegen.hugeMethodLimit` to avoid big method(> 8k) that can't be compiled by JIT compiler.

Data Sources

- Spark separates computation and storage.
- Complete data pipeline:
 - External storage feeds data to Spark.
 - Spark processes the data
- Data source can be a bottleneck if Spark processes data very fast.

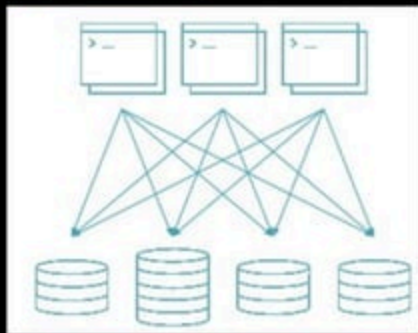
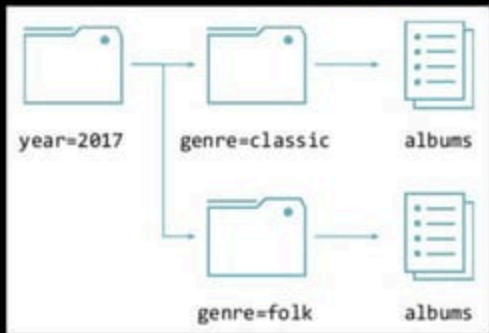
Scan Vectorization

- More efficient to read columnar data with vectorization.
- More likely for JVM to generate SIMD instructions.
-



Partitioning and Bucketing

- A special file system layout for data skipping and pre-shuffle.
- Can speed up query a lot by avoid unnecessary IO and shuffle.
- The summit talk: <http://dbricks.co/2oG6ZBL>



Performance Tips

- Pick data sources that supports vectorized reading. (parquet, orc)
- For file-based data sources, creating partitioning/bucketing if possible.

Yet challenges still remain

Raw Data

ETL

Data Lake

Analytics

Insight

Big data pipelines / ETL
Multiple data sources
Batch & streaming data

Machine Learning / AI
Real-time / streaming analytics
(Complex) SQL analytics

Reliability and Complexity

- Data corruption issues and broken pipelines
- Complex workarounds - tedious scheduling and multiple jobs/staging tables
- Many use cases require updates to existing data - not supported by Spark / Data lakes

Reliability & Performance Problems

- Performance degradation at scale for advanced analytics
- Stale and unreliable data slows analytic decisions

Databricks Delta address these challenges



Reliability & Automation

- Transactions guarantees eliminates complexity
- Schema enforcement to ensure clean data
- Upserts/Updates/Deletes to manage data changes
- Seamlessly support streaming and batch

Performance & Reliability

- Automatic indexing & caching
- Fresh data for advanced analytics
- Automated performance tuning

Thank you

Xiao Li (lixiao@databricks.com)

Wenchen Fan (wenchen@databricks.com)