Improving Spark's Reliability with DataSourceV2

Ryan Blue
Spark Summit 2019

NETFLIX

Data at Netflix

Cloud-native data warehouse

- YARN compute clusters are expendable
- Expendable clusters require architectural changes
  - **GENIE** is a job submission service that selects the cluster
  - **METACAT** is a cluster-independent metastore
  - **S3** is the source of truth for data

S3 is eventually consistent

- File list calls may be inaccurate
- Hive tables rely on accurate listing for correctness
- S3 queries may be incorrect, sometimes

S3 is eventually consistent

- File list calls may be inaccurate
- Hive tables rely on accurate listing for correctness
- S3 queries may be incorrect, **sometimes**

At Netflix's scale,
sometimes is every day.

A reliable S3 warehouse (in 2016)

- Requires consistent listing – **S3MPER**
- Requires in-place writes – **BATCH PATTERN**
- Requires atomic metastore changes – **METACAT**

Changes needed in Spark

- Integrate S3 batch pattern committers
- Spark versions
  - **1.6** – Hive path only
  - **2.0** – DataSource path for reads, not writes
  - **2.1+** – Use DataSource path for reads and writes

Problems and Roadblocks

DataFrameWriter

- Behavior is not defined
- What do `save` and `saveAsTable` do differently?
    - Create different logical plans . . .

      that are converted to other logical plans
- When you use “overwrite” mode, what happens?
    - Depends on the data source

SaveMode

- Delegates behavior to the source when tables don’t exist
- Overwrite might mean:
  - Replace table – data and metadata (Some code paths)
  - Replace all table data (Some code paths)
  - Replace static partitions (DataSource tables)
  - Replace dynamic partitions (Hive tables, SPARK-20236)

Validation

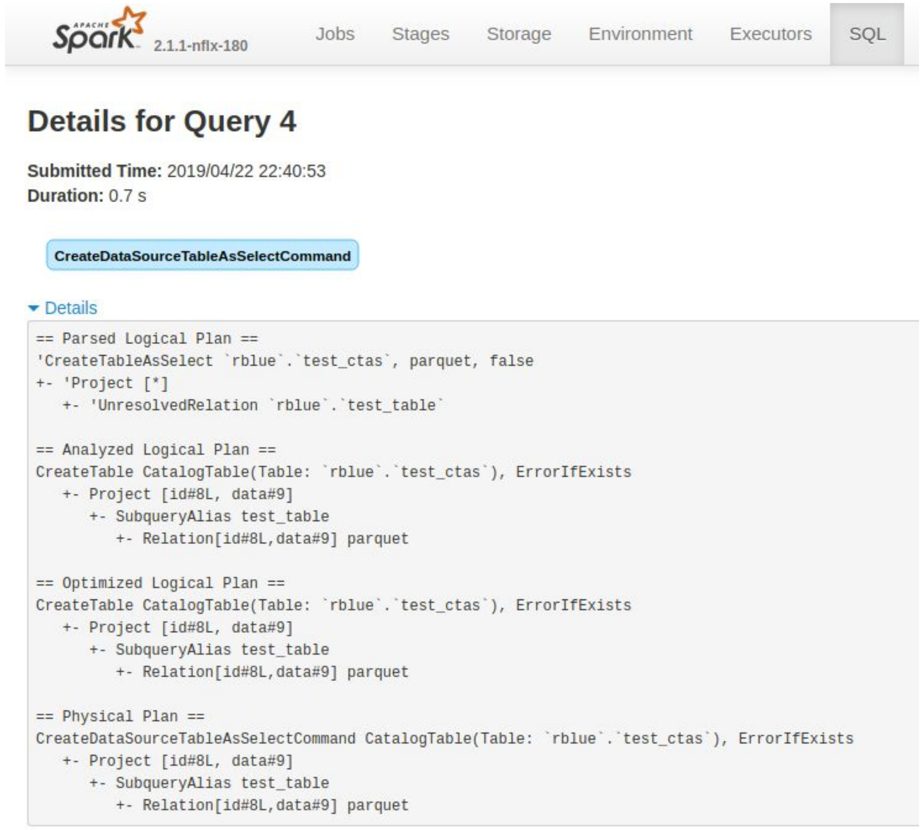
- What is “correct” for CTAS/overwrite when the table exists?
- `PreprocessTableCreation` vs `PreprocessTableInsertion`
    - Depends on the DataFrameWriter call
- Spark automatically inserts unsafe casts (e.g. string to int)
- Path tables have no schema validation on write

“[These] should do the same thing, but as we've already published these 2 interfaces and the implementations may have different logic, we have to keep these 2 different commands.”

“[These] should do the same thing, but as we've already published these 2 interfaces and the implementations may have different logic, we have to keep these 2 different commands.”

Commands

- `RunnableCommand` wraps a logical in a pseudo-physical plan
- Commands created inside `run` made it worse



Details for Query 4

Submitted Time: 2019/04/22 22:40:53
Duration: 0.7 s

```
== Parsed Logical Plan ==
'CreateTableAsSelect `rblue`.`test_ctas`, parquet, false
+- 'Project [*]
   +- 'UnresolvedRelation `rblue`.`test_table`

== Analyzed Logical Plan ==
CreateTable CatalogTable(Table: `rblue`.`test_ctas`), ErrorIfExists
   +- Project [id#8L, data#9]
      +- SubqueryAlias test_table
         +- Relation[id#8L,data#9] parquet

== Optimized Logical Plan ==
CreateTable CatalogTable(Table: `rblue`.`test_ctas`), ErrorIfExists
   +- Project [id#8L, data#9]
      +- SubqueryAlias test_table
         +- Relation[id#8L,data#9] parquet

== Physical Plan ==
CreateDataSourceTableAsSelectCommand CatalogTable(Table: `rblue`.`test_ctas`), ErrorIfExists
   +- Project [id#8L, data#9]
      +- SubqueryAlias test_table
         +- Relation[id#8L,data#9] parquet
```

Community Roadblocks

- Substantial behavior changes for 2.0
  - Committed with no time to review

    . . . to the 2.0 release branch
- Behavior not up for discussion
- Parts of PRs merged without attribution

Iceberg and DataSourceV2

A reliable S3 warehouse (in 2019)

- **Iceberg:** tables without unpleasant surprises
- Fix tables, not the file system

- While fixing reliability and scale, fix usability:
    - Reliable schema evolution
    - Automatic partitioning
    - Configure tables, not jobs

Last year

- Need a way to plug in Iceberg cleanly
- Maintaining a separate write path takes time
- Spark’s write path had solidified
- DataSourceV2 was proposed . . .

Why DataSourceV2?

- Isn’t v2 just an update to the read/write API?
- Existing design problems also affect v2
  - No write validation – yet another logical plan
  - SaveMode passed to sources
- **Opportunity**: avoid needing v3 to fix behavior

What's different in DSv2

- Define a set of common logical plans
  - CTAS, RTAS, Append, OverwriteByExpression, etc.
  - Document user expectations and behavior
  - Implement consistent behavior in Spark for all v2 sources

- SPIP: Standardize SQL logical plans
  https://issues.apache.org/jira/browse/SPARK-23521

Standard Logical Plans

- Specialize physical plans, not logical plans
  - No more `InsertIntoDataSourceTable` and `InsertIntoHiveTable`
  - No forgetting to apply rules to a new logical plan
- Apply validation rules universally
  - Same rules for Append and Overwrite
- Avoid using RunnableCommand

Consistent behavior

- Create, alter, and drop tables in Spark, not sources
  - CTAS when table exists: fail the query in Spark
  - Requires a catalog plugin API

- SPIP: Spark API for Table Metadata

  https://issues.apache.org/jira/browse/SPARK-27067

Catalog API

- Multi-catalog support
    - Create tables in the source of truth
    - Avoiding this caused strange Spark behavior

- SPIP: Identifiers for multi-catalog support
  https://issues.apache.org/jira/browse/SPARK-27066

Status

- **Goal:** working DSv2 in Spark 3.0
  - Independent of the v1 path
  - Default behavior to v1
- SPIPs have been adopted by community votes
- Append and overwrite plans are added and working
- Waiting on catalog API to add CTAS and DDL

Thank you!
Questions?

NETFLIX

Up next: **Migrating to Spark at Netflix**
At **11:50 today**, in **Room 2006**