A Smarter Pig: Building a SQL interface to Pig using Apache Calcite

Eli Levine & Julian Hyde

Apache: Big Data, Miami
2017/05/16

About us

Eli Levine @teleturn

PMC member of Phoenix
ASF member

Julian Hyde @julianhyde

Original developer of Calcite
PMC member of Calcite, Drill, Eagle, Kylin
ASF member

Apache Calcite

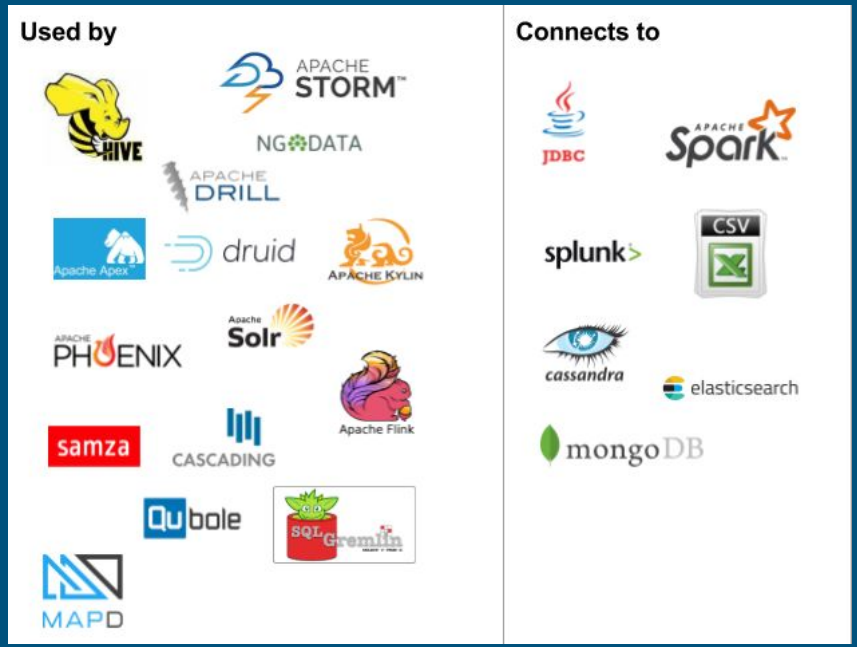
Apache top-level project since October, 2015

Query planning framework

- Relational algebra, rewrite rules
- Cost model & statistics
- Federation via adapters
- Extensible

Packaging

- Library
- Optional SQL parser, JDBC server
- Community-authored rules, adapters



Apache Pig

Apache top-level project

Platform for Analyzing Large Datasets

- Uses Pig Latin language
  - Relational operators (join, filter)
  - Functional operators (map, reduce)
- Runs as MapReduce (also Tez)
- ETL
- Extensible
  - LOAD/STORE
  - UDFs

Outline

Batch compute on Force.com Platform (Eli Levine)

Apache Calcite deep dive (Julian Hyde)

Building Pig adapter for Calcite (Eli Levine)

Q&A

Salesforce Platform

Object-relational data model in the cloud

Contains standard objects that users can customize or add their own

SQL-like query language SOQL

- Real-time
- Batch/ETL

```
SELECT DEPT.NAME
FROM EMPLOYEE
WHERE FIRST_NAME = ‘Eli’
```

Federated data store: Oracle, HBase, external

User queries span data sources (federated joins)

Salesforce Platform - Current Batch/ETL

Called Async SOQL

- REST API
- Users supply SOQL and info about where to deposit results

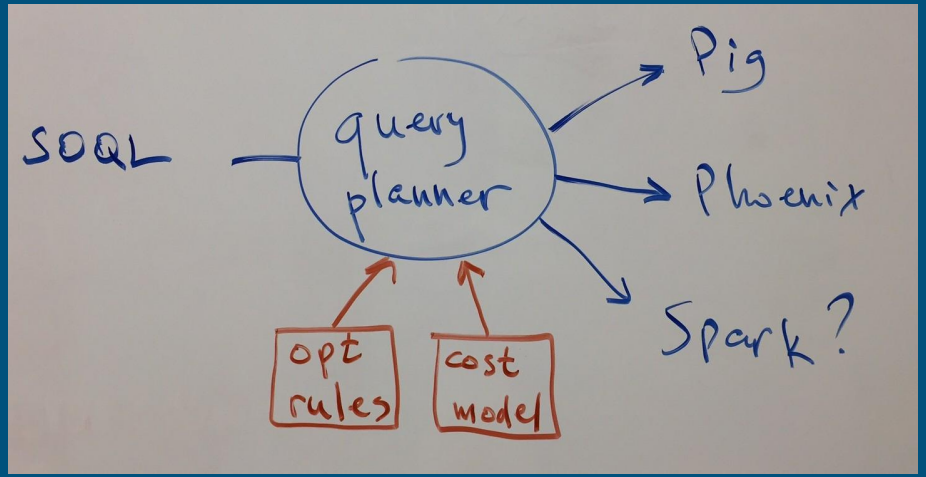
SOQL -> Pig Latin script

Pig loaders move data/computation to HDFS for federated query execution

Own SOQL parsing, no Calcite

Query Planning in Async SOQL

Current



Next generation

Apache Calcite for Next-Gen Optimizer

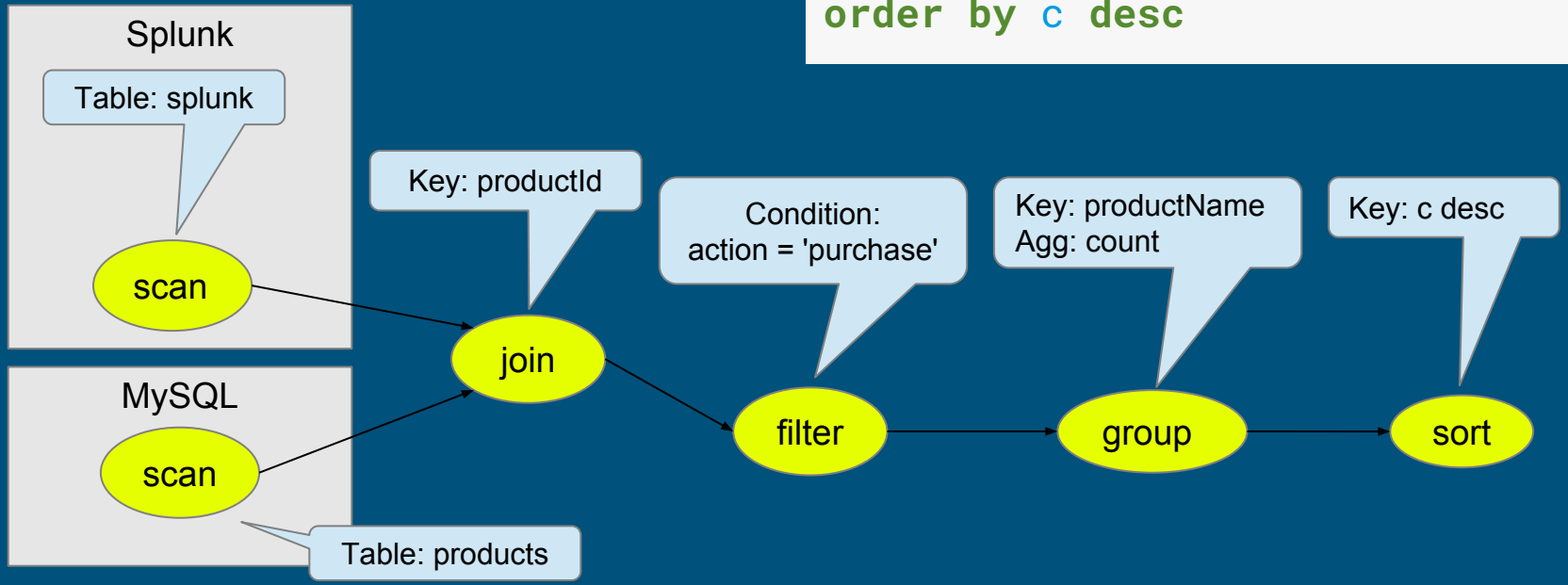
- Strong relational algebra foundation
- Support for different physical engines
- Pluggable cost model
- Optimization rules
- Federation-aware

[Julian talks about Calcite]

- Building blocks for building query engine or DB
- Federation-aware
- Represent and optimize logical plan
- Convert to physical plan

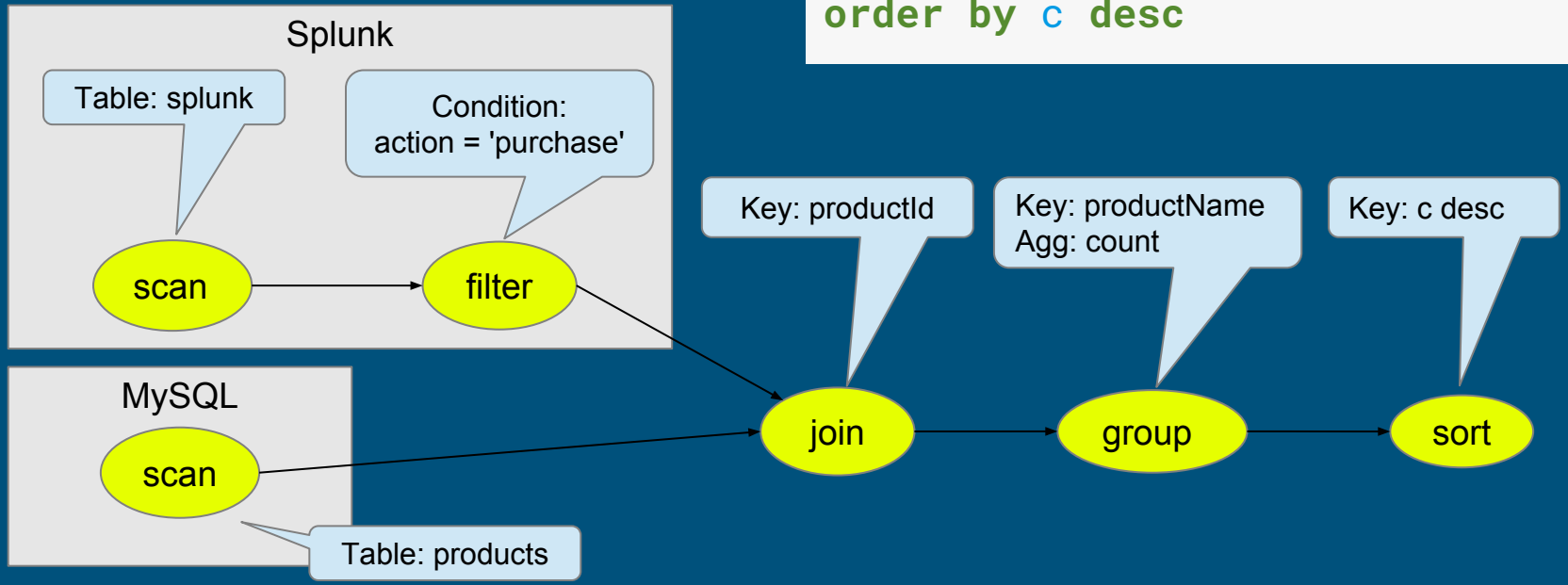
Planning queries

```
select p.productName, count(*) as c
from splunk.splunk as s
    join mysql.products as p
    on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```



Optimized query

```
select p.productName, count(*) as c
from splunk.splunk as s
    join mysql.products as p
    on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```



Adapters

Connect to a data source

How to push down logic to the data source?

A set of planner rules

Maybe also a calling convention

Maybe also a query model & query generator

Calcite Pig Adapter

```
SELECT DEPT_ID FROM EMPLOYEE GROUP BY DEPT_ID HAVING COUNT(DEPT_ID) > 10
```

```
EMPLOYEE = LOAD 'EMPLOYEE' ... ;
EMPLOYEE = GROUP EMPLOYEE BY (DEPT_ID);
EMPLOYEE = FOREACH EMPLOYEE GENERATE COUNT(EMPLOYEE.DEPT_ID) as DEPT_ID__COUNT_,
    group as DEPT_ID;
EMPLOYEE = FILTER EMPLOYEE BY (DEPT_ID__COUNT_ > 10);
```

Building the Pig Adapter

1. Implement Pig-specific RelNodes. e.g. PigFilter
2. RelNode Factories
3. Various RelOptRule types for converting Abstract RelNodes to PigRelNodes
4. Schema implementation (several iterations)
5. Unit tests run local Pig

Lessons Learned

Calcite is very flexible (both good and bad)

Lots available out of the box

Little info on writing optimization and conversion rules

Dynamic code generation using Janino -- cryptic errors

SQL maps well to Pig. Inverse not always true.

Thank you!

Eli Levine @teleturn
Julian Hyde @julianhyde
http://calcite.apache.org
http://pig.apache.org