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

Eli Levine & Julian Hyde

Apache: Big Data, Miami
2017/05/17

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

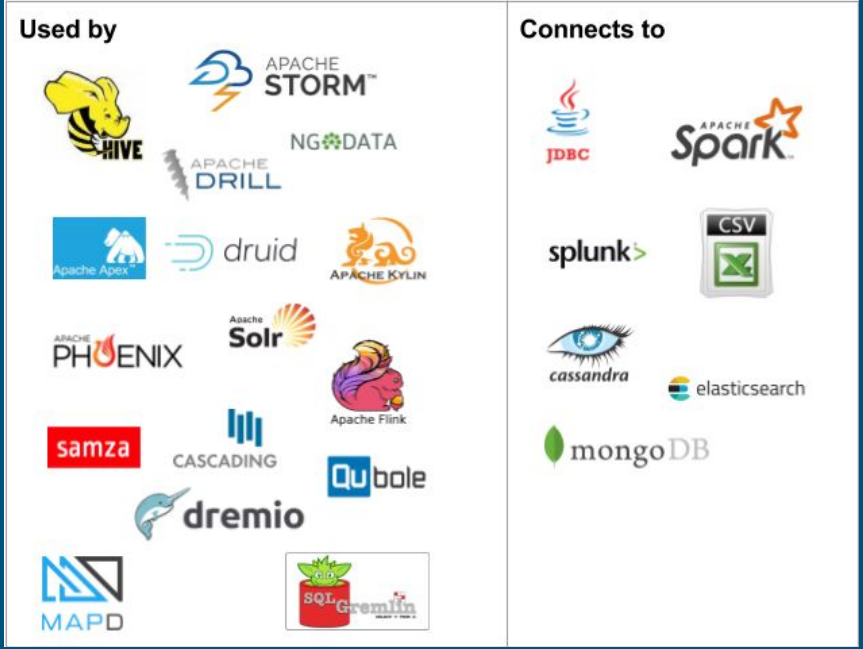
Apache top-level project since October, 2015

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 (mapreduce)
- ➢ 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 compute

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

Federated data store: Oracle, HBase, external

User queries span data sources (federated joins)

Salesforce Platform - Batch Compute

Called Async SOQL

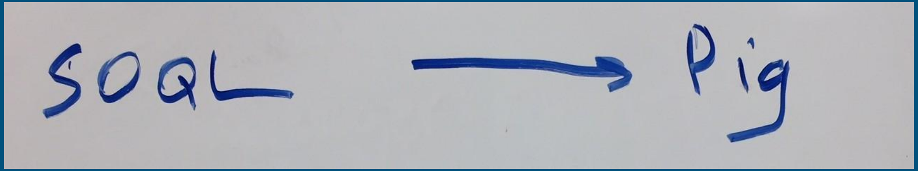
- REST API
- Users supply SOQL and info about where to deposit results

SOQL -> Pig Latin script

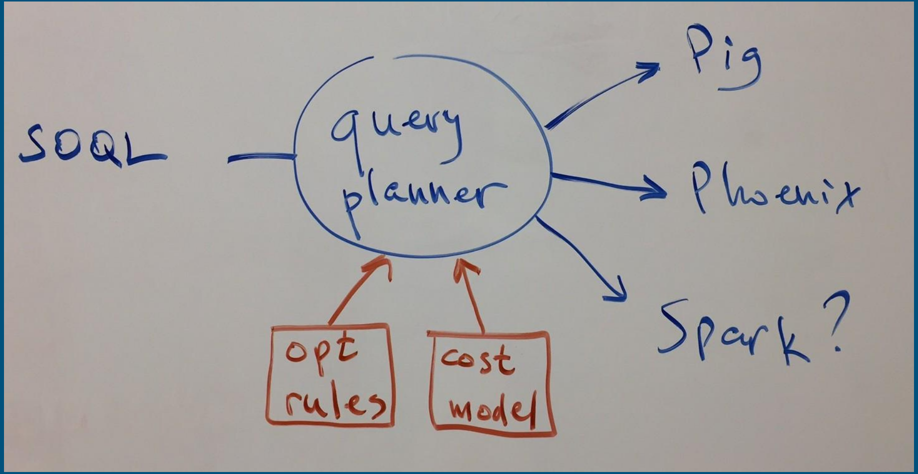
Pig loaders move data/computation to HDFS for federated query execution

Own SOQL parsing, no Calcite

Query Planning in Async SOQL



Current



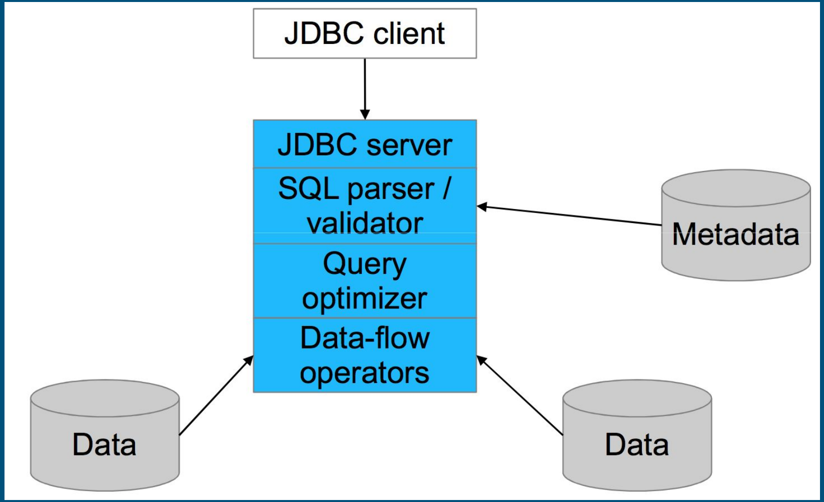
Next generation

Apache Calcite for Next-Gen Optimizer

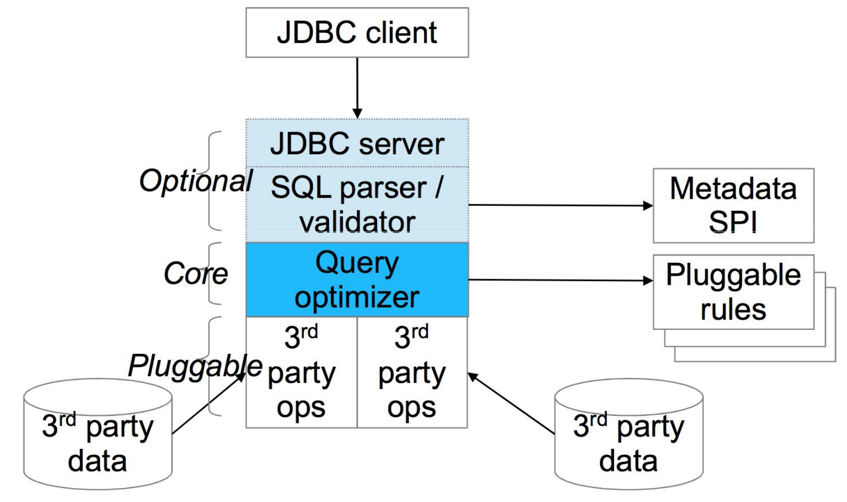
- Strong relational algebra foundation
- Support for different physical engines
- Pluggable cost model
- Optimization rules
- Federation-aware

Architecture

Conventional database



Calcite



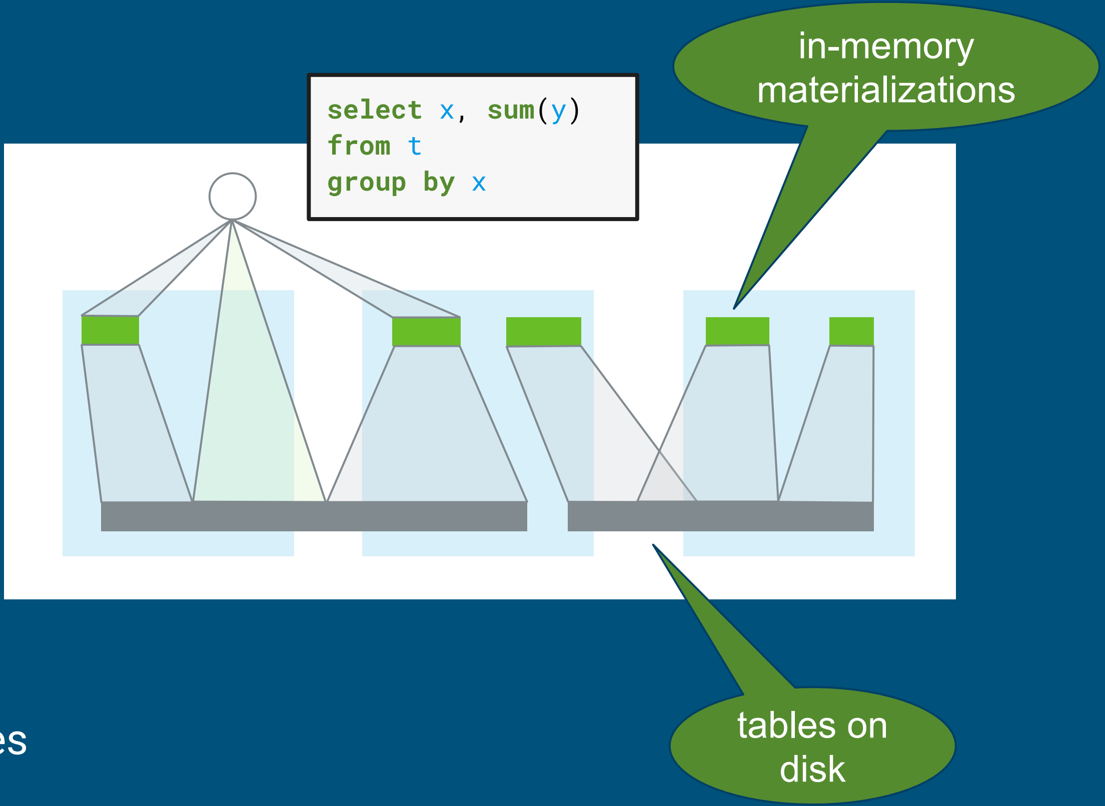
Calcite design

Design goals:

- Not-just-SQL front end
- Federation
- Extensibility
- Caching / hybrid storage
- Materialized views

Design points:

- Relational algebra
- Composable transformation rules



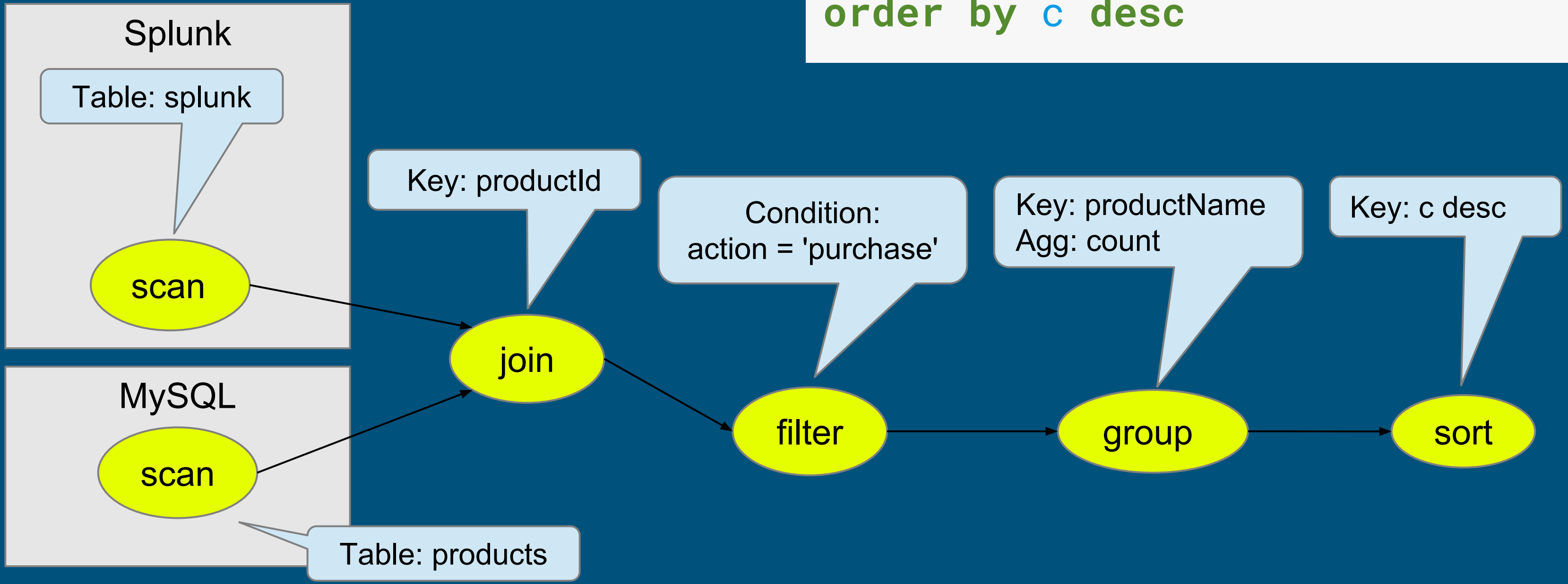
```
select x, sum(y)
from t
group by x
```

in-memory materializations

tables on disk

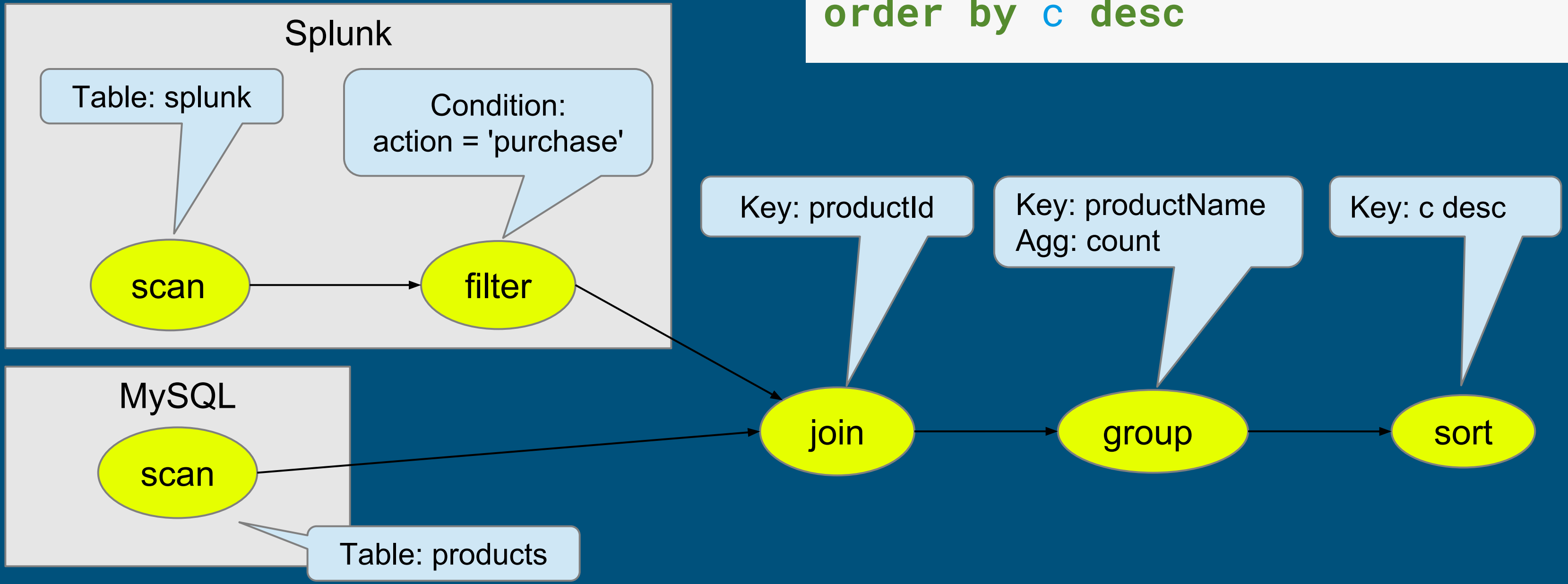
Planning queries

```
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

```sql
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
```



Calcite framework

Relational algebra

RelNode (operator)

- TableScan
- Filter
- Project
- Union
- Aggregate
- …

RelDataType (type)

RexNode (expression)

RelTrait (physical property)

- RelConvention (calling-convention)
- RelCollation (sortedness)
- RelDistribution (partitioning)

RelBuilder

SQL parser

SqlNode

SqlParser

SqlValidator

Metadata

Schema

Table

Function

- TableFunction
- TableMacro

Lattice

JDBC driver

Transformation rules

RelOptRule

- FilterMergeRule
- AggregateUnionTransposeRule
- 100+ more

Global transformations

- Unification (materialized view)
- Column trimming
- De-correlation

Cost, statistics

RelOptCost

RelOptCostFactory

RelMetadataProvider

- RelMdColumnUniquensss
- RelMdDistinctRowCount
- RelMdSelectivity

Adapter

Implement `SchemaFactory` interface

Connect to a data source using parameters

Extract schema - return a list of tables

Push down processing to the data source:

- A set of planner rules
- Calling convention (optional)
- Query model & query generator (optional)

```
"schemas": [
 {
   "name": "HR",
   "type": "custom",
   "factory":
"org.apache.calcite.adapter.file.FileSchemaFactory",
   "operand": {
     "directory": "hr-csv"
   }
 }
]

$ ls -l hr-csv
-rw-r--r--  1 jhyde  staff   62 Mar 29 12:57 DEPTS.csv
-rw-r--r--  1 jhyde  staff  262 Mar 29 12:57 EMPS.csv.gz
$ ./sqlline  -u jdbc:calcite:model=hr.json -n scott -p tiger
sqlline> select count(*) as c from emp;
'C'
'5'
1 row selected (0.135 seconds)
```

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. Write RelOptRules for converting abstract RelNodes to Pig RelNodes
4. Schema implementation
5. Unit tests run local Pig

Lessons Learned

Calcite is very flexible (both good and bad)

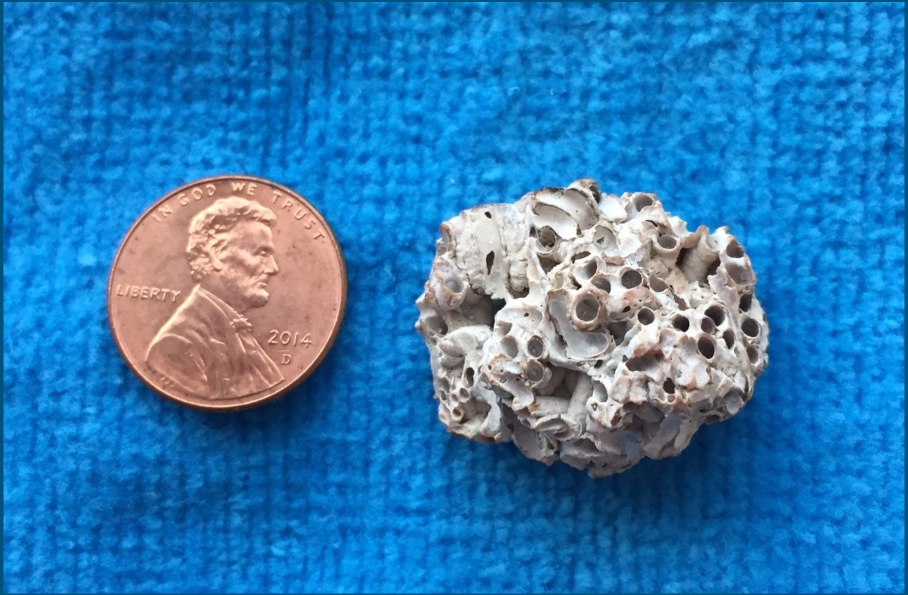
- “Recipe list” would be useful
- Lots of examples if you delve into existing adapters - e.g. Druid and Cassandra

Lots available out of the box

Dynamic code generation using Janino -- cryptic errors

RelBuilder was really useful (if you are building non-SQL engine)

Florida Calcite



Thank you!

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

APACHE:
BIG_DATA
NORTH_AMERICA

THE APACHE SOFTWARE FOUNDATION

Apache Calcite

salesforce

HORTONWORKS