Relational Beam:
Automatically optimize your pipeline

Andrew Pilloud
https://s.apache.org/beam-relational-2021
Agenda

1. What is Relational?
2. How can we optimize?
3. Today: Beam SQL
4. Tomorrow: Relational Beam
Beam is falling behind!
Beam is falling behind!

- Beam model has been mostly stable since 2015.
  - Schemas came out of SQL in 2018.
- What is the next big thing?
Beam is falling behind!

- Beam model has been mostly stable since 2015.
  - Schemas came out of SQL in 2018.
- What is the next big thing?

**Relational in Beam Core**

- The underlying runners have many of these features... since 2015!
What is Relational?
What is Relational?

- Relational Processing involves focusing on similarities among pieces of information
- Relational Optimization involves taking advantage of these similarities to reduce work
- Think traditional relational databases: Postgres, Oracle
(Traditional) Beam is not Relational

- Beam processes opaque records
  - Internally represented as byte[] or Object
  - Object form provided for user convenience
- Sometimes it processes <byte[] key, byte[] value>
  - Structure still opaque, only aware of equality
- Beam focuses on item-specific information
Beam is not Relational
Your data is Relational
Beam can be Relational

● We need metadata about the structure of your data
  ○ What is the structure of that byte[]?
  ○ How much data can we expect?

● We need metadata about the computations performed
  ○ What columns do you access?
  ○ What transforms are performed?
Beam Schemas enable Relational

```
Schema.builder()
    .addInt64Field("foo").addInt32Field("baz").build();
```

- Beam Schemas expose the structure of your data
  - This can often be inferred!
- Provides an abstraction on of data access (Row)
- Doesn’t provide metadata about computations
Beam SchemaIO enables Relational

SchemaIO from(String location, Row configuration, @Nullable Schema dataSchema);

public interface SchemaIO {
    PTransform<PBegin, PCollection<Row>> buildReader();
}

public interface PushdownProjector {
    PTransform<? extends PInput, PCollection<Row>> withProjectionPushdown(FieldAccessDescriptor);
}

- Beam SchemaIO exposes the structure of your IOs
- Doesn't provide metadata... yet.
Beam SQL is Relational

```
SELECT SUM(foo) AS baz, end_of_window
FROM my_topic WHERE something_is_true(bizzle)
GROUP BY TUMBLING(timestamp, 1 HOUR)
HAVING baz > my_magic_number LIMIT 3;
```

- Relational model: Projection, Filter, Aggregation
- ... and advanced bits like nested ROW, ARRAY, UNNEST
- Optimizations only within SqlTransform
Java Schema Transforms are Relational Too!

my_topic
  .apply(Select.fieldNames("foo", "end_of_window"))

● Not all operators generate metadata for optimization
● No optimizations yet
How can we optimize?
Global Relational Optimizer

- Allow a pipeline to be transformed after expand
  - Eventually optimizing portability protos
- No core model for this yet!
  - Where does the optimizer run?
- Beam Java design mailed Tuesday!
Global Relational Optimizer

TableScan → Project → Join → Filter → TableModify

TableScan → Project → Join → Filter → TableModify

TableScan → Filter → Project → Join → TableModify

TableScan → Filter → Project → Join → TableModify
Column Pruning

- Stop passing unused fields as soon as possible
  - Ideally at the source IO but also before shuffles
- Beam Java has a model for this: FieldAccessDescriptor
  - PTransform provides a list of accessed columns
- Beam Java has a new implementation on Schema IO!
Column Pruning
Join Algorithm Selection and Reordering

- Automatically choose optimal joins
  - Also reorder joins
- No core model for this yet!
  - Need an interface to query IOs for statistics
- Beam SQL has an implementation
Join Algorithm Selection and Reordering

Stream → Bounded → SideInput → SideInput → TableModify

Stream → Bounded → CoGroup → SideInput → TableModify
Row Expression

- Calcite calls this a RexNode
  - `SELECT <row>` and `WHERE <bool>` from SQL

- Three Required Operators
  - Field Access (FieldAccessDescriptor)
  - Constant (Schema Value)
  - Call (Arbitrary function call, the difficult one)
Filter Pushdown

- Apply filters as early as possible
  - Ideally at the source IO but also before shuffles
- No core model for this yet!
  - Need a “row expression” language
- Beam SQL has an implementation
Filter Pushdown

TableScan → Project → Join → Filter → TableModify

TableScan → Project → Join → Project → FilterScan → TableModify

FilterScan → Project → Join → Project → FilterScan → TableModify
Project Pushdown

- Stop passing unused data as soon as possible
  - Ideally at the source IO but also before shuffles
- Beam Java’s FieldAccessDescriptor may be extended
  - Need a “row expression” language
- Beam SQL has an implementation but no IO support
Filter and Project Pushdown

Diagram showing the process of TableScan, Project, Join, Filter, and TableModify operations, with a comparison of the pre-pushdown and post-pushdown stages.
Row Expression Execution

- Allow the optimizer to decide how to execute
  - Eventually pushed down to Runner
- No core model for this yet!
  - Need a “row expression” language
- Beam SQL has multiple implementations
Row Expression Execution

Java

```java
input.apply(
    SqlTransform.query(sql))
```

SQL (via Java)

```sql
SELECT key, a + b + c
FROM input WHERE d > 3
```
Row Expression Execution

Java

```
input.apply(
    SqlTransform.query(sql))
```

SQL (via Java)

```
SELECT key, a + b + c
FROM input WHERE d > 3
```
Vectorized Execution

for (i) { z[i] = x[i] + y[i] }

- Structure data in memory for efficient execution
  - Requires batches, Benefits unclear for Streaming
- No core model for this yet!
  - Java 16 may only require internal changes
- Beam Dataframes has an implementation
Columnar Coders

- Structure data in transit for efficient execution
  - Requires batches, Benefits unclear for Streaming
- No core model for this yet!
  - May only require internal changes
- Apache Arrow as a coder
Zero-Copy Project

- Fields can be projected without deserialization or copy
  - Benefit for columnar fields
  - Also for large or expensive streaming fields
- No core model for this yet!
  - May only require internal changes
Deferred Deserialization

- Don’t deserialize fields until first access
  - Benefit for large or expensive fields
- No core model for this yet!
  - May only require internal changes
Order Aware Pcollections

● Some attribute of the data is ordered
  ○ Could be time, could be another key
● No core model for this yet!
Retractions

- Sometimes your data is actually a change log!
- Beam is “append only” today.
  - What about a delete?
  - What about a change?
- No core model for this yet!
  - How will it work with IOs
Today: Beam SQL
SqlTransform No Longer Experimental!

- As of Beam 2.33.0 (Coming late September)
  - [https://github.com/apache/beam/pull/15244](https://github.com/apache/beam/pull/15244)
Beam SQL: It’s Apache Calcite, essentially.

- SQL Parsing and Validation*
- Conversion to Relational Algebra*
- Conversion to Physical Execution Plan
- JDBC Driver
- Implementation of Built-in SQL operators
- Project and Filter Code Generation
Apache Calcite

Parse to AST

Validate AST

Convert to Logical Plan

Convert to Physical Plan

SELECT key, SUM(value)
FROM input GROUP BY key

Beam Java

input.apply(SqlTransform.query(sql))
SQL Parsing, Validation, and Conversion

● Apache Calcite handles this
  ○ We’ve extended the parser to support our DDL syntax
  ○ We provide Calcite with schemas

● Outputs abstract Relational Algebra model of SQL (tree of RelNodes)
  ○ Filter
  ○ Project
  ○ Join
  ○ Aggregate
  ○ Values
  ○ TableScan* (BeamIOSource)
  ○ TableModify* (BeamIOSink)
  ○ ...
The Calcite Model

RelNode

Query

TableScan

Filter

Project

Join

Agg

TableModify

TableScan

Filter

Project

TableModify
The Beam Model

IO Source → ParDo → ParDo → CoGroup → Combine → IO Sink

IO Source → ParDo → ParDo → IO Sink

PTransform

PCollection (bounded or unbounded)
**SQL Conversion to Physical plan**

- We use Calcite’s implementation of the Volcano Optimizer
  - Uses Rules to convert to a Physical plan and costs to optimize
- Calcite provides basic rules to simplify the RelNodes
  - Filter + Project = Calc
- Beam provides physical RelNodes and rules
  - Calc -> BeamCalc
  - Join -> BeamJoin
  - Aggregate -> BeamAggregation
  - Values -> BeamValues
  - BeamEnumerableConverter*
  - ...
- Beam RelNodes are PTransforms
Beam Calc (Expression Evaluator)

- Beam Calc is a simple ParDo operation in Beam
- Wraps Calcite reference implementation of EnumerableCalc
  - Starting in Beam 2.10, prior versions used an interpreter
  - Generates Java code for operators at pipeline creation time
  - Complete support for Calcite built-in project functions
- Also have ZetaSQL Calc wrapping ZetaSQL’s reference implementation
  - Relatively slow due to cost of calling from Java to C++
Apache Calcite Code Generation

- Generates Java code for row expressions

  SELECT id, convert(price), price * 10 WHERE item = "my item" ...

  Becomes

  ```java
doFn(Context c, Row r) {
    if ("my item".equals(r.get(2))) {
      int price = r.get(1);
      c.output(new Row(r.get(0),
                      MyUdf.convert(price), price * 10));
    }
  }
```
ZetaSQL

- ZetaSQL == BigQuery Standard SQL
- Written in C++, currently only works on (modern) Linux systems
- Currently Parses and Validates SQL
- Basic support in Beam with ZetaSQL SqlTransform
- Does not replace Calcite!
- Still @Experimental
Tomorrow: Relational Beam
Relational Beam needs Schemas

● Beam Schemas expose the structure of your data
● Schema Row further abstracts the data
  ○ Enables some optimizations without user changes
  ○ Required for now
● Not using Schemas?
  ○ You Get Nothing! You lose! Good day, sir!
Relational Beam needs SchemaIO

- Schema IO is a standardized (internal) interface to IOs
  - Can be retrofitted into existing IOs
  - Not a replacement for builders
- We are still adding the Relational pieces
  - Project and Filter Push-down
  - Record Count and Rate Statistics
Relational Beam needs Field Access Descriptor

- Use Schema Transforms
- Use SqlTransform
- Annotate your Java ParDos with @FieldAccess
- Eventually Static analysis?
Relational Beam wants More!

- Use high level interfaces when possible
  - Schema Transforms
  - SqlTransform
  - Dataframes
  - More?
Relational Beam: Automatically optimize your pipeline

Andrew Pilloud / apilloud@apache.org

These Slides - https://s.apache.org/beam-relational-2021