Peloton Cost Model

Alex Steere
Gus Angulo
Nate Appelson

Starting Goals

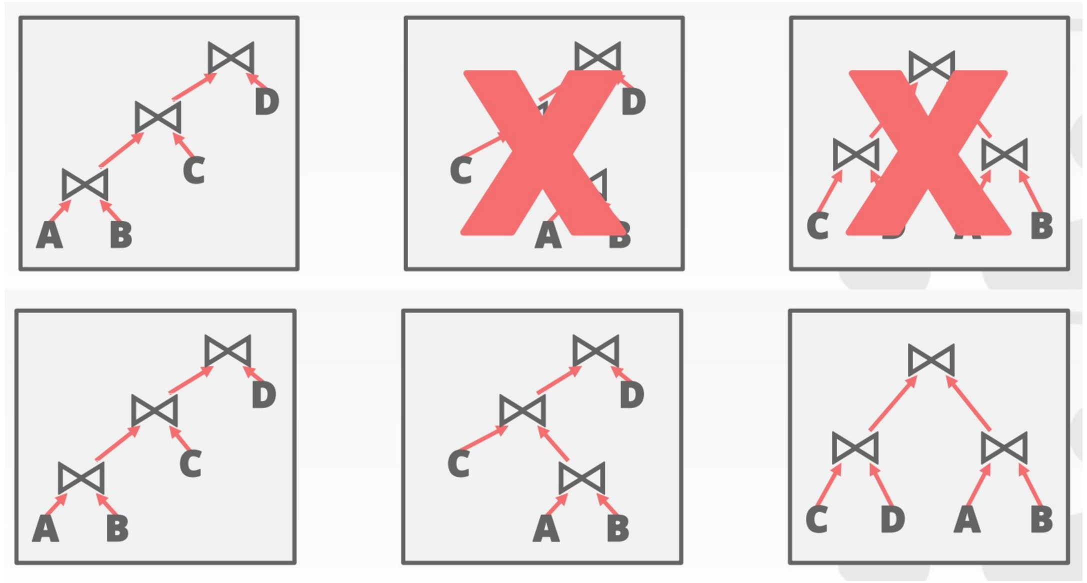
Establish support for optimal join reordering in the cost model

Create testing infrastructure for the optimizer+cost model

Peloton now supports join-reordering in the optimizer

Before

After



Change of plans

We realized that the cost model was in worse shape than we thought

When we started, the optimizer would pick the ordering based on the order of tables in the query or the wrong order

No point doing optimal join ordering if we’re not even picking correct orderings

Update on Goals

Peloton now supports join reordering in optimizer

Implemented a Postgres style cost-model for hash joins

Added testing infrastructure foundation to make it easier to evaluate cost models

Many refactors and bug-fixes as the existing optimizer and cost model had issues that needed to be addressed first

Postgres Hash Join Cost Model Motivation

SELECT * FROM A INNER JOIN B WHERE A.v = B.b;

<u>Nested Loop Join</u>

```
O(|A|)  for x in A:
O(|B|)    for y in B:
            If x == y: emit()
```

<u>Hash Join</u>

```
                  table = []

O(|B|)            For x in B:
                    table.insert(x)

O(|A|)            for x in A:
O(|bucket|)         If table.find(x): emit()
```

Postgres Hash Join Cost Model

$$|ProbeTable| \cdot BucketSize$$

The bucket size is an estimate of the expected number of tuples to each hash bucket

It takes into account:

- Low cardinality skew
- High frequency skew

Testing Infrastructure

- Allows users to specify
  - Cost model
  - Data loading sql file
  - List of sql queries
  - Number of times to perform the queries
- Launches peloton, loads data, runs queries, returns simple statistics and output of optimizer
- Python based and uses psycopg2

Demo

Refactors and additional improvements

Refactored optimizer to take in cost model as argument into constructor.

Added cost model settings flag so users can specify cost model of choice when running peloton

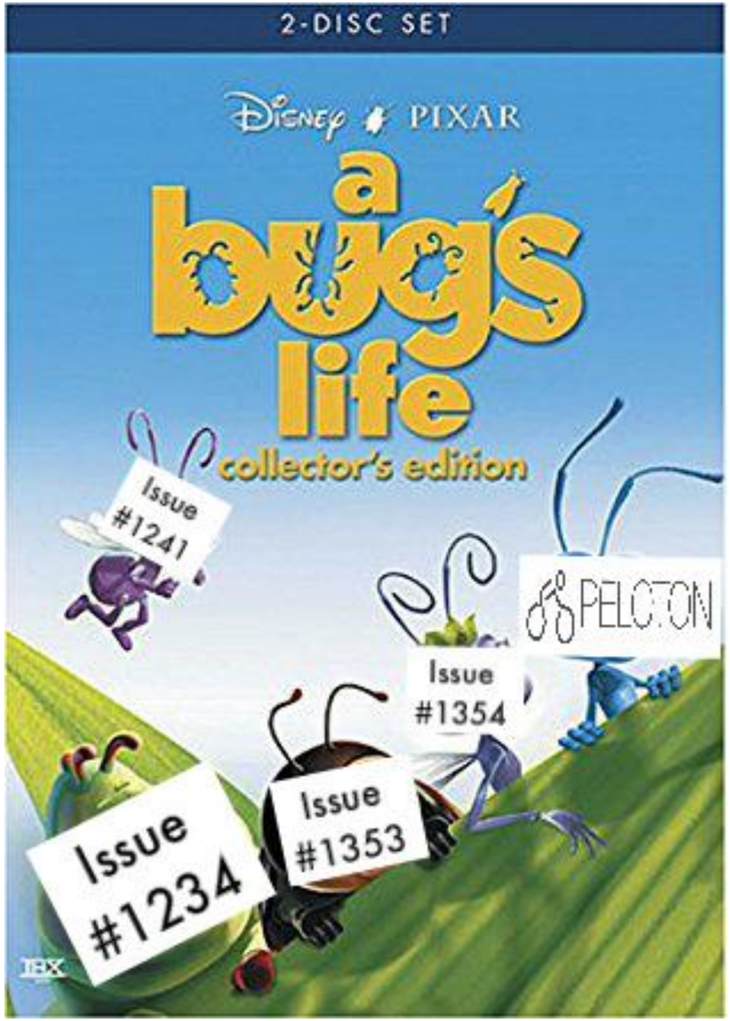
Testing Strategies

- Simple c++ plan selection tests
  - Join ordering
  - Join type (e.g. hash vs nested-loop)
  - Different size tables
- Testing framework
  - Evaluate differences between existing cost model and “Postgres” cost-model

Bug Fixes

We discovered many bugs when we started working, which unfortunately took up a lot of time to fix

- Column stats not being generated
- Costs being doubled or tripled
- Transformation rules not being applied
- Equality between sub-plans not working properly



Setbacks Along the Way

We did not expect the issues in the optimizer and cost model

Many bugs in the optimizer required fixing

Need to manually invoke analyze (makes it harder to perform benchmarks)

Future Work

- Further research into cost model techniques
  - Learned models
  - Further mimicking other open source versions (e.g. Postgres)
- More feature rich testing infrastructure
  - Integration with existing benchmarks
  - Ability to easily compare many different cost models
  - Further statistics and plan introspection
  - Integration with other DBMSes

Questions?