Low Latency Query Planning and Processing

Philipp Fent
Hardware gets fast

- Large main memory
- Fast SSDs
- Many core machines

Low latency queries with fast data access

- Now bound by CPU capabilities
- Algorithmic changes
Algorithmic challenges

Query processing
- Intra-query parallelism
- Shared state

Query planning
- Cardinality estimation
- Algebra operations
Groupjoin – Idea

- Combined execution of compatible join and aggregation
- Q: “Total sales per customer”

```
join = Hashtable()
for c in customer:
    join[c.id] = c

group = Hashtable()
for o in orders:
    if join.contains(o.c_id):
        group[o.c_id].sum += o.price

groupjoin = Hashtable()
for c in customer:
    groupjoin[c.id] = c

for o in orders:
    if groupjoin.contains(o.c_id):
        groupjoin[o.c_id].sum += o.price
```
Groupjoin – Avoiding contention

- Shared hash table unsuitable for multithreaded execution
- Four execution strategies for parallel groupjoin

\[
\Gamma_{\text{C.id, sum(0.price)}} \\
\text{C.id} = 0.\text{c_id}
\]

Customer Orders

Separate

Eager

Memoizing

Index

Global Hashtable

Thread-local Hashtables

\[
\begin{align*}
T_1 & : r_1 = \{3 : \gamma_1\} \\
T_2 & : r_2 = \{3 : \gamma_2\} \\
T_3 & : r_3 = \{
\end{align*}
\]
Nested Aggregates

- Common in analytical queries
- HAVING predicates are hard to estimate

Q: “Customer with large orders”

```sql
select l_orderkey
from lineitem
group by l_orderkey
having sum(l_quantity) > 300
```
Nested Aggregates

- Common in analytical queries
- HAVING predicates are hard to estimate
- But have significant impact on execution plans

![Diagram a) Selective \( \sigma \)-Predicate](image1)
![Diagram b) Unselective \( \sigma \)-Predicate](image2)
Estimating Aggregates

- Numerical columns \( \sim N(\mu, \sigma^2) \)
- Cheap and generalizes nicely, but inherently symmetric
Estimating Aggregates

- Using a skew-normal distribution
Practical Groupjoins and Nested Aggregates

- Effects ⅛ of queries
- +23% in TPC-H, +6% in TPC-DS

![Bar chart for TPC-H and TPC-DS showing speedup for various queries.](chart.png)
Query Optimization with Indexed Algebra

- Complex queries on small workloads
  - BigQuery: 90% of queries processed less than 100 MB of data
  - Tableau Public: 90% of workbooks are less than 100k tuples

- TPC-H
  - Scale 1: 0.8 ms optimization, 20 ms execution
  - Scale 0.01: 0.8 ms optimization, 0.2 ms execution

- Optimization time scales super-linear with query complexity
Algebra

- Relational algebra trees
  - Operators
  - Expressions
  - Columns / IUs

- Analyze data-flow for optimization
  - Which path?
  - Modifications?
  - Materialized?
Optimization

- Reason about the algebra to derive optimization possibilities
- Top-down, operator at a time
  - Needs $O(n^2)$ column sets
- Path-centric
  - Still $O(n)$ length
  - With indexing: $O(\log n)$
Indexing Algebra

- Index paths through the algebra
  - Faster path traversal
- Binary search trees on path depth
- Paths from root might overlap

(a) Represented algebra plan
(b) Balanced binary index of the path from B\(^6\) to the root
(c) Index from D\(^5\) to the root
Indexing Algebra

- Index paths through the algebra
  - Faster path traversal
- Binary search trees on path depth
- Paths from root might overlap
- Link/cut trees support that efficiently
Indexing Algebra

- Index paths through the algebra
  ➡ Faster path traversal
- Binary search trees on path depth
- Paths from root might overlap
- **Link/cut trees** support that efficiently

<table>
<thead>
<tr>
<th></th>
<th>Rel. Algebra</th>
<th>Transformation</th>
<th>Traversal</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o index</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td></td>
</tr>
<tr>
<td>static index</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td></td>
</tr>
<tr>
<td>path labeling</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Indexed Algebra</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td></td>
</tr>
</tbody>
</table>
Indexed Algebra Performance

- Significant overall improvements
- 10 - 30% faster optimization
- 8% better *end-to-end* latency in Tableau Public
Conclusion

Query processing
✅ Intra-query parallelism
✅ Shared state

Query planning
✅ Cardinality estimation
✅ Algebra operations