Special Topics:
Self-Driving Database Management Systems
Automatic SQL Rewriting

@Jia Qi Dong // 15-799 // Spring 2022
LAST CLASS (DATAFARM)

- Generate abstract plans through Markov Chains
- Instantiate them based on real workload’s distribution
TODAY’S AGENDA

Overview
MCTS
Deep Rewrite Estimation
Experiments
ARCHITECTURE OVERVIEW

1. **SQL Query**
   - **SQL Rewriter**
     - (Optional / Rare)

2. **SQL Query**
   - **Parser**

3. **Abstract Syntax Tree**
   - **Binder**

4. **Logical Plan**
   - **Tree Rewriter**
     - (Optional / Common)

5. **Logical Plan**
   - **Optimizer**

6. **Physical Plan**

**SOURCE:** CMU 15645 – QUERY PLANNING PART 1
MOTIVATION

Web, Java, Tablet

Performance Regression

Database

Bad SQLs

Try out logical transformations

SOURCE: XUANHE ZHOU
MOTIVATION

Web, Java, Tablet  Performance Regression  Database  Bad SQLs  Try out logical transformations

SELECT MAX(DISTINCT L1.col1) FROM lineitem L1 WHERE L1.col1 = ANY (SELECT MAX(C.col1) m_key FROM customer C, lineitem L2 WHERE C.col1 = L2.col1 AND ((C.col2<2 AND C.col3<2) OR (C.col2<2 AND L2.col2>5)) GROUP BY C.col1);

Execution Time > 20min

SELECT MAX(L1.col1) FROM lineitem L1, customer C WHERE L1.col1 = C.col1 AND (C.col2<2 OR (C.col3<2 AND L2.col2>5))

Over 600x ↑

SELECT MAX(L1.col1) FROM lineitem L1, customer C WHERE L1.col1 = C.col1 AND (C.col2<2 OR (C.col3<2 AND L2.col2>5))

Execution Time > 1.941s

SOURCE: XUANHE ZHOU
REWRITE RULES

➢ Remove redundant aggregates

```sql
select max(distinct a) from t; →
select max(a) from t;
```

➢ Create temporary table

```sql
select * from t1 where a1 < any(select a2 from t2);
→
With t as (select a2 a from t2)
selct t1.* from t1, t where a1<a
```
EXISTING APPROACHES

➢ DBAs rewrite queries based on some **rules** and **past experiences**
  ➢ **Problem:** can’t generalize to a large number of queries
  ➢ Can’t decide which rules are better

➢ Heuristic query rewrite (Postgres)
  ➢ **Problem:** may find local optimum instead of global optimum
Input Query

```
"SELECT MAX ( DISTINCT L1.col1 )
FROM lineitem L1
WHERE L1.col1 = ANY
    ( SELECT MAX
        ( C.col1 ) m_key
        FROM customer C,
        lineitem L2
        WHERE C.col1 = L2.col1
        AND ((
            C.col2<2
        AND C.col3<2 )
        OR ( C.col2<2
        AND L2.col2>5 ))
GROUP BY C.col1);"
```
TOP-DOWN ORDER (POSTGRES)
TOP-DOWN ORDER (POSTGRES)
Cannot further optimize temporary table
Execution: > 20 min
OPTIMAL ORDER

Query Tree

Rewrite in Optimal Order
Rewritten Query

```
SELECT MAX(L1.col1)
FROM lineitem L1, customer C
WHERE L1.col1 = C.col1
  AND (C.col2 < 2 OR (C.col3 < 2 AND L2.col2 > 5));
```

Execution: 1.941 s
CHALLENGES

➢ Goal: Transform a SQL query into an equivalent query with minimal execution time

➢ Challenges:
   ➢ How to represent large amount of possible ordering?
     ➢ 10 rules → 30,000 different rewritten queries
   ➢ How to find optimal ordering of rules efficiently?
     ➢ Rewrite requires low overhead (milliseconds)
     ➢ Need a light-weight model/alg
   ➢ How to estimate benefit of whole rewrite order?
     ➢ Rewrite one operator may affect benefit of others
Today’s Agenda

Overview
MCTS
Deep Rewrite Estimation
Experiments
MONTE CARLO TREE SEARCH

- RL algorithm commonly used in tree search
- AlphaGo
- Root = original query
- Child node is obtained by applying a rewrite rule to an operator
MONTE CARLO TREE SEARCH

➢ Node → a rewritten query
➢ Edge → (operator, rule) pair
➢ $C^\uparrow$ → cost reduction from $v_0$
   ➢ Cost estimated by optimizer
   ➢ $C^\uparrow(v_i) = \text{Cost}(v_0) - \text{Cost}(v_i)$
➢ $C^\downarrow$ → Subsequent cost
   ➢ Estimated using NN
   ➢ $C^\downarrow(v_i) = \text{Cost}(v_i) - \text{Cost}(v^*_i)$
➢ $F$ → Access frequency

\[
U(v_i) = (C^\uparrow(v_i) + C^\downarrow(v_i)) + \gamma \sqrt{\frac{ln(F(v_0))}{F(v_i)}}
\]
1. Selection
   → V2 is selected
MONTE CARLO TREE SEARCH

1. Selection

2. Expansion
   For each operator o:
      For each rule r:
         if \((o,r)\) appliable:
            add child node
MONTE CARLO TREE SEARCH

1. Selection

2. Expansion

3. $C\downarrow$ Estimation

$\rightarrow C\downarrow(v2) = 0.1$
MONTE CARLO TREE SEARCH

1. Selection

2. Expansion

3. $C\downarrow$ Estimation

4. Utility Update
   → If ancestor $v'$ has smaller benefit, update $C\downarrow(v')$
   → Increase $F(v')$
   → Update $U(v')$
PARALLEL QUERY REWRITE

- Select **multiple nodes** to expand at each iteration
- The nodes do NOT have ancestor-descendant relationships
- Sum of their utilities is maximized
- Dynamic Programming!
TODAY’S AGENDA

Overview
MCTS
Deep Rewrite Estimation
Experiments
CHALLENGES

➢ Goal → Estimate C↓
➢ Challenges
  ➢ Many factors affecting cost reduction (query operator, rules, data distribution …)
  ➢ Rewrite rules are correlated (applying a rule make other rules unapplicable)
  ➢ Expensive to obtain labelled training data
MULTI-HEAD ATTENTION MODEL

Feature Encoding
- Rule Encoding
- Query Encoding
- Metadata Encoding

Rule Embedding
- Multi-head Attention
- FC

Rule Selection
- Normalization
- FC

Cost Estimation
- Multi-head Attention
- FC

$h^0 = H^0 \times n \times 2k$

$h^1 = H^1 \times n \times 2k$

$h^2 = H^2 \times 1 \times 2k$

$C_{+}(q)$
**RULE ENCODING**

- What is the cost of each (operator, rule) pair?
- $M^R[i, j] = \text{cost reduction of applying rule } i \text{ on operator } j$
- $M^R[i, j] = 0$ if not applicable
- Estimated by the optimizer
QUERY ENCODING

① Feature Encoding

Rule Encoding

\[ M^R \]

<table>
<thead>
<tr>
<th>rule (n)</th>
<th>0.01</th>
<th>0</th>
<th>0</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>5.6e+3</td>
<td>3.1e+6</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>5.6e+3</td>
<td>0</td>
<td>...</td>
<td>5.5e+3</td>
</tr>
</tbody>
</table>

Query Encoding

\[ M^Q \]

<table>
<thead>
<tr>
<th>col 1</th>
<th>1</th>
<th>1</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>col k</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Metadata Encoding

\[ M^D \]

<table>
<thead>
<tr>
<th>index</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>distinct ratio</td>
<td>0.82</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

➢ What columns does each operator reference?
➢ \( M^Q[i, j] = 1 \) if operator \( j \) contains column \( i \)
➢ \( M^Q[i, j] = 0 \) otherwise
### METADATA ENCODING

#### ① Feature Encoding

**Rule Encoding**

\[
M^R = \begin{bmatrix}
0.01 & 0 & 0 & \ldots & 0 \\
0 & 5.6e+5 & 3.1e+6 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 5.6e+5 & 0 & \ldots & 5.5e+5
\end{bmatrix}
\]

**Query Encoding**

\[
M^Q = \begin{bmatrix}
1 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0
\end{bmatrix}
\]

**Metadata Encoding**

\[
M^D = \begin{bmatrix}
1 & 1 & 1 & 0 \\
0.82 & 1 & 1 & 1
\end{bmatrix}
\]

- Does column i have index?
- \( M^D[0, j] = 1 \) if column j has index
- How many distinct values does each column have?
- \( M^D[1, j] = \) distinct value ratio of column j
① Feature Encoding

Rule Encoding

$M^R$ operator (m)

<table>
<thead>
<tr>
<th>0.01</th>
<th>0</th>
<th>0</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.6e+5</td>
<td>3.1e+6</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

rule (n)

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.6e+5</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

Query Encoding

$M^Q$ operator (m)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>col 1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>col k</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Metadata Encoding

$M^D$ column (k)

<table>
<thead>
<tr>
<th>index</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>distinct ratio</td>
<td>0.82</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
COMBINED ENCODING

1. Feature Encoding

Rule Encoding

\[ M^R \]

- operator (m)
- rule (n)

<table>
<thead>
<tr>
<th>0.01</th>
<th>0</th>
<th>0</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.6e+5</td>
<td>3.1e+6</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5.6e+5</td>
<td>0</td>
<td>...</td>
<td>3.8e+5</td>
</tr>
</tbody>
</table>

Query Encoding

\[ M^Q \]

- operator (m)
- col 1
- ... 
- col k

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Metadata Encoding

\[ M^D \]

- column (k)
- index
- distinct ratio

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.82</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Does this operator have column j, and does column j have index?
- Yes

- Does this operator have column j, and what is its distinct ratio?
- No

- M operators

- k columns

- k columns
Attention = Similarity

Compute similarity of each rule (row) wrt other rules (other rows)

Similar rules

affect the same operators by the same amount

Likely to cause rewrite conflict

\[ H^0'[i, \ast] = H^0[i, \ast] + \sum_{j \neq i} \alpha_{i,j} H^0[j, \ast] \]
RULE SELECTION

➢ $H^1$ has $n$ rows $\rightarrow$ rewrite benefits of all rules
➢ $H^2$ has 1 row $\rightarrow$ represents rule with optimal benefits
COST ESTIMATION

Rule Embedding

Multi-head Attention

\[ H^0_n \times m \]

\[ M^{Q+D}_{m \times 2k} \]

Rule Selection

Normalization

\[ H^1 \]

\[ H^1' \]

\[ H^2 \]

Cost Estimation

Multi-head Attention

\[ C^+(q) \]

15-799 Special Topics (Spring 2022)

CMU-DB
TRAINING DATA GENERATION

- Randomly assemble \{Table, Join, Predicate, Aggregate Operation, Column\}
- Use SQLSmith to synthesize queries via SQL syntax tree
- Cluster queries (DBSCAN) based on cost vectors
- Sample 5% queries to enumerate optimal rewrite costs
- Take avg cost reduction as label for queries in the cluster
Many noises in training data since labels are based on small part of queries

\[ L(q) = \left( F(q) - C \downarrow (q) \right)^2 + \sum_{i,j} \mu_{i,j} |F(q_i) - F(q_j)| \]

Minimize MSE between estimation model and labelled cost reduction

Minimize L1 distance between queries in the same cluster as they should have similar cost reduction
TODAY’S AGENDA

Overview
MCTS
Deep Rewrite Estimation
Experiments
EXPERIMENT SETTING

➢ **Machine**: 16GB RAM, 256GB disk, 4.00GHz CPU, Titan RTX 2080Ti GPU with 11 GB buffer

➢ **Dataset**:
  ➢ TPC-H 1x (~4.7G), TPC-H50x (~50G)
  ➢ JOB (~1.1G) → OLAP with IMDB data, 16,000 queries
  ➢ XuetangX (~11.5G) → OLTP benchmark, 22,000 queries

➢ **Metrics**
  ➢ **Execution cost**: from query optimizer
  ➢ **Rewrite latency**: time of rewriting a query
  ➢ **Query Latency**: time of answering a query
  ➢ **Overall Query Latency**: rewrite latency + query latency
EXECUTION COST AND QUERY LATENCY

- **Baseline**
  - TopdownPostgres
  - TopdownCalcite
  - Heuristic → Always select the rule with most benefit
  - Arbitrary → randomly select operators

- **Tree Search + cost estimation**
  = better rewrite orders

*Figure 5: Performance Comparison on 1G TPC-H Queries.*

*Figure 6: Performance Comparison on JOB Queries.*

*Figure 7: Performance Comparison on XuetangX Queries.*
LearnedRewrite has high rewrite latency, but lowest overall latency

<table>
<thead>
<tr>
<th>Method</th>
<th>Rewrite Latency</th>
<th>Query Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary</td>
<td>3.3 - 10.1 ms</td>
<td>553.2 s</td>
</tr>
<tr>
<td>TopdownPostgres</td>
<td>0.3 - 3.9 ms</td>
<td>427.5 s</td>
</tr>
<tr>
<td>TopdownCalcite</td>
<td>1.5 - 18.9 ms</td>
<td>431.1 s</td>
</tr>
<tr>
<td>Heuristic</td>
<td>5.8 - 24.2 ms</td>
<td>331.7 s</td>
</tr>
<tr>
<td>LearnedRewrite</td>
<td>6.1 - 69.8 ms</td>
<td><strong>224.5 s</strong></td>
</tr>
</tbody>
</table>
EXPLORATION PARAM

- $\uparrow \gamma \rightarrow \uparrow$ rewrite latency; $\downarrow$ query latency
- Sweet spot at $\gamma = 1.4 \times 10^5$
- Use higher $\gamma$ for slow queries
  - Main bottleneck is query latency
- lower $\gamma$ for fast queries
  - Reduce rewrite overhead
TREE SEARCH ALGO

➢ Baseline: DFS, B(est)FS

Figure 11: Comparison of Tree Search Algorithms (TPC-H).
ESTIMATION MODEL

- Tested against 2 strategies:
  - **Rewrite(E):** Enumerate all rewrite orders
  - **Rewrite(S):** Sample K rewrite orders

- LearnedRewrite produces faster queries
PARALLEL REWRITE

(a) Rewrite Latency

(b) Overall Latency

Figure 10: Comparison of Parallel Algorithms (TPC-H). DP denotes DP-based parallel rewrite; TopK denotes greedy parallel rewrite; and Single denotes selecting single node.
LearnedRewrite’s model can adapt to different rule combinations (pad unused rules as 0)

Can also efficiently search in large search space

Figure 12: Adaptability on Different Number of Rules.
VARYING NUMBER OF OPERATORS

(a) TPC-H

(b) JOB
PRROBLEMS

➢ Retrain if new tables/columns
➢ Similarity = rewrite conflict?
➢ All queries in the cluster have the same label?

➢ Demo