A Layered Aggregate Engine for Analytics Workloads

Maximilian Schleich
University of Oxford

Dan Olteanu, University of Oxford
Mahmoud Abo Khamis, relationalAI
Hung Q. Ngo, relationalAI
XuanLong Nguyen, University of Michigan

University of Washington
July, 2019
Recall relationalAI Keynote: Analytics over Databases

Current State of Affairs in Analytics Workloads

- Carefully crafted by domain experts
- Throws away relational structure
- Comes with relational structure
- Can be order-of-magnitude larger
Turn Analytics Workload into Database Workload!

Database Workload: **Batches of Aggregate Queries**

**Advantages:**

1. Use DB Tools for Optimization
2. Decompose Aggregates into Views over Join Tree
   - Pushing aggregate computation past joins
   - Using different roots and directional views
3. Avoid Materialization of Data Matrix

**Challenge:**

- Workloads require **many** aggregate queries
## Aggregates are at the Core of Analytics Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>Query Batch</th>
<th># Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>( \text{SUM}(X_i \times X_j) )</td>
<td>140</td>
</tr>
</tbody>
</table>
| Covariance Matrix       | \( \text{SUM}(X_i) \text{ GROUP BY } X_j \)  \
|                         | \( \text{COUNT}(*) \text{ GROUP BY } X_i, X_j \) |           |
| Regression Tree (1 Node)| \( \text{VARIANCE}(Y) \text{ WHERE } X_j = c_j \) | 270       |
| Mutual Information      | \( \text{COUNT}(*) \text{ GROUP BY } X_i \) | 106       |
| Chow-Liu Trees          | \( \text{COUNT}(*) \text{ GROUP BY } X_i, X_j \) |           |
| Data Cubes              | \( \text{SUM}(M) \text{ GROUP BY } X_1, \ldots, X_d \) | 40        |

(# Queries shown for Favorita Kaggle dataset)
Existing DBMSs are **NOT** Designed for Query Batches

Relative Speedup for *Our Approach* over DBX and MonetDB

C = Covariance Matrix;  R = Regression Tree Node;  AWS d2.xlarge (4 vCPUs, 32GB)
Tools of a Database Researcher

1. Exploit structure in the data
   ▶ Algebraic structure: Factorized aggregate computation
   ▶ Combinatorial structure: Query complexity measures

2. Sharing computation and data access
   ▶ Aggregates decomposed into views over join tree
   ▶ Share data access across views

3. Specialization for workload and data
   ▶ Generate code specific to the query batch and dataset
   ▶ Improve cache locality for hot data

4. Parallelization
   ▶ Task and domain parallelism
LMFAO: Layered Multi Functional Aggregate Optimization

App → LMFAO

- Application
- Aggregates
- Join Tree

Logical Optimization

- Merge Views
- Aggregate Pushdown
- Find Roots

Code Optimization

- Group Views
- Multi-Output Optimization
- Parallelization
- Compilation
The Layers of LMFAO: Logical Optimization

\[ Q_1: \text{SUM (units)} \]
\[ Q_2: \text{SUM (item} \cdot f(\text{date, color})) \text{ GROUP BY store} \]
\[ Q_3: \text{SUM (units} \cdot \text{item)} \text{ GROUP BY color} \]

Favorita Kaggle Dataset:
Units sold for different items, stores, date.
The Layers of LMFAQO: Logical Optimization

\[ Q_1: \text{SUM}(\text{units}) \]
\[ Q_2: \text{SUM}(\text{item} \cdot f(\text{date}, \text{color})) \quad \text{GROUP BY} \ \text{store} \]
\[ Q_3: \text{SUM}(\text{units} \cdot \text{item}) \quad \text{GROUP BY} \ \text{color} \]

**Find Roots Layer:**
For each query, decide its output (root) node. Choose root which minimizes sizes of views.
The Layers of LMFAO: Logical Optimization

\[ Q_1: \text{SUM(units)} \]
\[ Q_2: \text{SUM}(\text{item} \cdot f(\text{date}, \text{color})) \quad \text{GROUP BY store} \]
\[ Q_3: \text{SUM}(\text{units} \cdot \text{item}) \quad \text{GROUP BY color} \]

**Aggregate Pushdown Layer:**

Break down each query into *directional views* over the join tree.

Reuse Partial Aggregates & **Merge Views** with same group-by attributes.
The Layers of LMFAO: Code Optimization

\[ Q_1: \text{SUM}(\text{units}) \]
\[ Q_2: \text{SUM}(\text{item} \cdot f(\text{date}, \text{color})) \quad \text{GROUP BY store} \]
\[ Q_3: \text{SUM}(\text{units} \cdot \text{item}) \quad \text{GROUP BY color} \]

**Group Views Layer:**

1. Construct Dependency Graph
2. Group Views that are computed over same relation
The Layers of LMFAO: Code Optimization

\[ Q_1: \text{SUM (units)} \]
\[ Q_2: \text{SUM (item} \cdot f(\text{date, color})) \quad \text{GROUP BY store} \]
\[ Q_3: \text{SUM (units} \cdot \text{item)} \quad \text{GROUP BY color} \]

Multi-Output Optimization Layer:
View Group is a \textbf{computational unit} in LMFAO.
All views in one group are computed in one scan over the relation.
The Layers of LMFAO: Code Optimization

- \( Q_1: \text{SUM}(\text{units}) \)
- \( Q_2: \text{SUM}(\text{item} \cdot f(\text{date}, \text{color})) \) GROUP BY store
- \( Q_3: \text{SUM}(\text{units} \cdot \text{item}) \) GROUP BY color

**Parallelization Layer:**
- **Task parallelism:** Evaluate independent groups in parallel
- **Domain parallelism:** Partition the large relation used by each group

Diagram:
- Application
  - Aggregates
  - Join Tree
  - Find Roots
  - Aggregate Pushdown
  - Merge Views
  - Group Views
  - Multi-Output Optimization
  - Parallelization
  - Compilation
The Layers of LMFAO: Code Optimization

\[ Q_1: \text{SUM}(\text{units}) \]
\[ Q_2: \text{SUM}(\text{item} \cdot f(\text{date}, \text{color})) \quad \text{GROUP BY} \text{ store} \]
\[ Q_3: \text{SUM}(\text{units} \cdot \text{item}) \quad \text{GROUP BY} \text{ color} \]

Compilation Layer:
Generate C++ code to compute each View Group.
Code Generation for Executing View Group 6 over Sales

\[ Q_1: \text{SUM (units)} \]

Traverse Sales as a trie following an order of its join attributes
Code Generation for Executing View Group 6 over Sales

\[ V_I \rightarrow \text{item} \]

\[ \forall i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I \bowtie_{\text{item}} V'_I) \]

\[ V_H \rightarrow \text{date} \]

\[ \forall d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T) \]

\[ V_T \rightarrow \text{store} \]

\[ \forall s \in \pi_{\text{store}}(\sigma_{\text{item}=i, \text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T) \]

\[ Q_1: \text{SUM (units)} \]

Lookup into incoming views, e.g., \( V_H \), as early as possible
Code Generation for Executing View Group 6 over Sales

\[ V_I \rightarrow \text{item} \]
\[ V'_I \rightarrow \text{item} \]
\[ V_H \rightarrow \text{date} \]
\[ V_T \rightarrow \text{store} \]

\[ \alpha_0 = 0; \]
\[ \text{foreach } i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_I \bowtie_{\text{item}} V'_I) \]
\[ \alpha_1 = V_I(i) \]
\[ \alpha_3 = 0; \]
\[ \text{foreach } d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T) \]
\[ \alpha_4 = V_H(d); \]
\[ \alpha_6 = 0; \]
\[ \text{foreach } s \in \pi_{\text{store}}(\sigma_{\text{item}=i, \text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T) \]
\[ \alpha_8 = V_T(d, s); \]
\[ \alpha_9 = 0; \]
\[ \text{foreach } u \in \pi_{\text{units}}(\sigma_{\text{item}=i, \text{date}=d, \text{store}=s} S : \alpha_9 += u; \]
\[ \alpha_6 += \alpha_8 \cdot \alpha_9; \]
\[ \alpha_3 += \alpha_4 \cdot \alpha_6; \]
\[ \alpha_0 += \alpha_1 \cdot \alpha_3 \]
\[ Q_1 = \alpha_0; \]

\[ Q_1: \text{SUM (units)} \]
Insert code for partial aggregates as early as possible
Reduces number of executed instructions
Code Generation for Executing View Group 6 over Sales

\[ V_i \rightarrow \text{item} \]
\[ V_i' \rightarrow \text{item} \]
\[ V_H \rightarrow \text{date} \]
\[ V_T \rightarrow \text{store} \]

\[ \alpha_0 = 0; \]
\[ \text{foreach } i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V_i \bowtie_{\text{item}} V'_i) \]
\[ \alpha_1 = V_i(i) \]
\[ \alpha_2 = i; \]
\[ \alpha_3 = 0; \]
\[ \text{foreach } d \in \pi_{\text{date}}(\sigma_{\text{item}=i} S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T) \]
\[ \alpha_4 = V_H(d); \]
\[ \alpha_6 = 0; \]
\[ \text{foreach } s \in \pi_{\text{store}}(\sigma_{\text{item}=i, \text{date}=d} S \bowtie_{\text{store}} \sigma_{\text{date}=d} V_T) \]
\[ \alpha_8 = V_T(d, s); \quad \alpha_9 = 0; \]
\[ \text{foreach } u \in \pi_{\text{units}}(\sigma_{\text{item}=i, \text{date}=d, \text{store}=s} S : \alpha_9 + = u; \]
\[ \alpha_6 + = \alpha_8 \cdot \alpha_9; \]
\[ \alpha_3 + = \alpha_4 \cdot \alpha_6; \]
\[ \alpha_0 + = \alpha_1 \cdot \alpha_3 \quad V_{S\rightarrow I}(i) = \alpha_3 \cdot \alpha_2; \]
\[ Q_1 = \alpha_0; \]

\[ V_{S\rightarrow I}: \text{SUM (units} \cdot \text{item) GROUP BY item} \]

Different outputs share partial aggregates
Code Generation for Executing View Group 6 over Sales

<table>
<thead>
<tr>
<th>$V_i$</th>
<th>$V_i'$</th>
<th>$V_H$</th>
<th>$V_T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td><code>item</code></td>
<td><code>item</code></td>
<td><code>date</code></td>
<td><code>store</code></td>
</tr>
</tbody>
</table>

\[
\alpha_0 = 0; \\
\text{foreach } i \in \pi_{item}(S \bowtie_{item} V_i \bowtie_{item} V'_i) \\
\alpha_1 = V_i(i) \\
\alpha_2 = i; \\
\alpha_3 = 0; \\
\text{foreach } d \in \pi_{date}(\sigma_{item=i} S \bowtie_{date} V_H \bowtie_{date} V_T) \\
\alpha_4 = V_H(d); \quad \alpha_5 = 0; \\
\text{foreach } c \in \pi_{color} \sigma_{item=i} V'_i : \alpha_5 += f(d, c) \cdot V'_i(i, c); \\
\alpha_6 = 0; \quad \alpha_7 = \alpha_5 \cdot \alpha_2 \cdot \alpha_4; \\
\text{foreach } s \in \pi_{store}(\sigma_{item=i, date=d} S \bowtie_{store} \sigma_{date=d} V_T) \\
\alpha_8 = V_T(d, s); \quad \alpha_9 = 0; \quad \alpha_{10} = |\sigma_{item=i, date=d, store=s} S|; \\
\text{foreach } u \in \pi_{units} \sigma_{item=i, date=d, store=s} S : \alpha_9 += u; \\
\alpha_{6} += \alpha_8 \cdot \alpha_9; \quad \alpha_{11} = \alpha_7 \cdot \alpha_8 \cdot \alpha_{10}; \\
\text{if } Q_2(s) \text{ then } Q_2(s) += \alpha_{11} \text{ else } Q_2(s) = \alpha_{11}; \\
\alpha_3 += \alpha_4 \cdot \alpha_6; \\
\alpha_0 += \alpha_1 \cdot \alpha_3 \quad V_{S \rightarrow I}(i) = \alpha_3 \cdot \alpha_2; \\
\]

\[Q_2 : \text{SUM (item } \cdot f(\text{date, color})) \quad \text{GROUP BY store}\]

Different outputs share partial aggregates
Experimental Evaluation

Relative Speedup for LMFAO over TensorFlow and MADlib

With at least same accuracy!

L = Linear Regression;  R = Regression Tree;  C = Classification Tree;
TensorFlow learns only 1 Decision Tree Node.  Intel i7-4770 (8 CPUs, 32GB)