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

ACM SIGMOD

June, 2019
Relational Data is Ubiquitous

**Kaggle Survey:** Most Data Scientists use Relational Data at Work!

![Bar graph showing percentage of Relational, Text, Image, Video, and Other data types used by data scientists overall, with Relational at 65.5%, Text at 53%, Image at 18.1%, Video at 5.1%, and Other at 10.3%.](image1)

![Bar graph showing percentage of Relational, Text, Image, Video, and Other data types used by data scientists in different industries, with Retail at 86%, Marketing at 83%, Insurance at 82%, and Financial at 77%.](image2)

(based on 2017 Kaggle survey of 16,000 ML practitioners)
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!

Many analytics workloads require computation of

batches of aggregate queries.

Advantages:

1. Use DB tools for optimization
2. Decompose Aggregates into views over join tree
   ▶ Using different roots and directional views
   ▶ Pushing aggregate computation past joins
3. Avoid materialization of data matrix

Challenge:

1. Workloads require many aggregate queries

In contrast:

1. Many ML systems rely on Linear Algebra packages for optimizations
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><code>SUM(X_i*X_j)</code></td>
<td>814</td>
</tr>
<tr>
<td>Covariance Matrix</td>
<td><code>SUM(X_i)</code> <code>GROUP BY X_j</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>COUNT(*)</code> <code>GROUP BY X_i, X_j</code></td>
<td></td>
</tr>
<tr>
<td>Decision Tree (Regression, 1 Node)</td>
<td><code>VARIANCE(Y)</code> <code>WHERE X_j = c_j</code></td>
<td>3,141</td>
</tr>
<tr>
<td>Mutual Information</td>
<td><code>COUNT(*)</code> <code>GROUP BY X_i</code></td>
<td>56</td>
</tr>
<tr>
<td>Chow-Liu Trees</td>
<td><code>COUNT(*)</code> <code>GROUP BY X_i, X_j</code></td>
<td></td>
</tr>
<tr>
<td>Data Cubes</td>
<td><code>SUM(M)</code> <code>GROUP BY X_1, \ldots, X_d</code></td>
<td>40</td>
</tr>
</tbody>
</table>

(# Queries shown for Retailer 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

Logical Optimization

Application

Aggregates

Join Tree

Merge Views

Aggregate Pushdown

Find Roots

Group Views

Multi-Output Optimization

Parallelization

Compilation

Code Optimization
The Layers of LMFAO: Logical Optimization

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

**Favorita Kaggle Dataset:**
Units Sales for different store, date, item.
The Layers of LMFAO: Logical Optimization

$Q_1$: \text{SUM}(f(\text{units}))

$Q_2$: \text{SUM}(g(\text{item}) \cdot h(\text{date}, \text{color})) \text{ GROUP BY store}

$Q_3$: \text{SUM}(f(\text{units}) \cdot g(\text{item})) \text{ GROUP BY 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

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

Aggregate Pushdown Layer:
Break down each query into \textit{directional views} over the join tree.
Reuse Partial Aggregates & \textbf{Merge Views} with same group-by attributes.
The Layers of LMFAO: Code Optimization

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

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

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

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

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

**Parallelization Layer:**

Task parallelism: Evaluate independent groups in parallel
Domain parallelism: Partition the large relation used by each group
The Layers of LMFAO: Code Optimization

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

Compilation Layer:
Generate C++ code to compute each View Group.
$Q_1: \text{SUM}\ (f(\text{units}))$

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

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

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

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

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

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

Insert code for partial aggregates as early as possible
Reduces number of executed instructions
Code Generation for Executing View Group 6 over Sales

\( V_I \rightarrow \text{item} \)

\( \alpha_0 = 0; \)

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

\( \alpha_1 = V_I(i) \)

\( \alpha_2 = g(i); \)

\( \alpha_3 = 0; \)

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

\( \alpha_4 = V_H(d); \)

\( \alpha_6 = 0; \)

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

\( \alpha_8 = V_T(d, s); \)

\( \alpha_9 = 0; \)

\( \text{foreach } u \in \pi_{\text{units}} \sigma_{\text{item}=i, \text{date}=d, \text{store}=s} S : \alpha_9 \leftarrow f(u); \)

\( \alpha_6 \leftarrow \alpha_8 \cdot \alpha_9; \)

\( \alpha_3 \leftarrow \alpha_4 \cdot \alpha_6; \)

\( \alpha_0 \leftarrow \alpha_1 \cdot \alpha_3 \)

\( V_{S \rightarrow I}(i) = \alpha_3 \cdot \alpha_2; \)

\( Q_1 = \alpha_0; \)

\( V_{S \rightarrow I}: \text{SUM} \left( f(\text{units}) \cdot g(\text{item}) \right) \text{ GROUP BY item} \)

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

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

\[ \alpha_0 = 0; \]
\[ \text{foreach } i \in \pi_{\text{item}}(S \bowtie_{\text{item}} V \bowtie_{\text{item}} V'_{I}) \]
\[ \alpha_1 = V_I(i) \]
\[ \alpha_2 = g(i) ; \]
\[ \alpha_3 = 0 ; \]
\[ \text{foreach } d \in \pi_{\text{date}}(\sigma_{\text{item}=i}S \bowtie_{\text{date}} V_H \bowtie_{\text{date}} V_T) \]
\[ \alpha_4 = V_H(d) ; \quad \alpha_5 = 0 ; \]
\[ \text{foreach } c \in \pi_{\text{color}}\sigma_{\text{item}=i}V'_I : \quad \alpha_5 += h(d, c) \cdot V'_I(i, c) ; \]
\[ \alpha_6 = 0 ; \quad \alpha_7 = \alpha_5 \cdot \alpha_2 \cdot \alpha_4 ; \]
\[ \text{foreach } s \in \pi_{\text{store}}(\sigma_{\text{item}=i,\text{date}=d}S \bowtie_{\text{store}} \sigma_{\text{date}=d}V_T) \]
\[ \alpha_8 = V_T(d, s) ; \quad \alpha_9 = 0 ; \quad \alpha_{10} = |\sigma_{\text{item}=i,\text{date}=d,\text{store}=s}S| ; \]
\[ \text{foreach } u \in \pi_{\text{units}}\sigma_{\text{item}=i,\text{date}=d,\text{store}=s}S : \quad \alpha_9 += f(u) ; \]
\[ \alpha_6 += \alpha_8 \cdot \alpha_9 ; \quad \alpha_{11} = \alpha_7 \cdot \alpha_8 \cdot \alpha_{10} ; \]
\[ \text{if } Q_2(s) \text{ then } Q_2(s) += \alpha_{11} \text{ else } Q_2(s) = \alpha_{11} ; \]
\[ \alpha_3 += \alpha_4 \cdot \alpha_6 ; \]
\[ \alpha_0 += \alpha_1 \cdot \alpha_3 \quad V_{S \rightarrow I}(i) = \alpha_3 \cdot \alpha_2 ; \]
\[ Q_1 = \alpha_0 ; \]

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

Different outputs share partial aggregates
Experimental Evaluation

Relative Speedup for **LMFAO** over **TensorFlow** and **MADlib**

With at least same accuracy!

![Bar Chart]

$L = $ Linear Regression; $R = $ Regression Tree; $C = $ Classification Tree;

TensorFlow learns only 1 Decision Tree Node. Intel i7-4770 (8 CPUs, 32GB)