LINVIEW: Incremental View Maintenance for Complex Analytical Queries

Milos Nikolic, Mohammed El Seidy, Christoph Koch
DATA, EPFL

SIGMOD, 24th June 2014
Big Data Analytics

Simple (SQL) Analytics

- Data Warehouses (OLAP)

Complex (non-SQL) Analytics

- Machine Learning
- Scientific Computing
- Data Mining
Complex Analytical Queries

- Often expressed as linear algebra on array data

  Example: Ordinary Least Squares

\[
Y = X \beta
\]

\[
\beta^* = (X^T X)^{-1} X^T Y
\]

- Multidimensional arrays

  HIGH DIMENSIONAL: Data processing is increasingly expensive

  DYNAMIC: Continuously changing, evolve through small changes
  (e.g., user’s Internet activity)

- Users want frequently fresh views of data

Re-evaluating complex queries on every (small) change is inefficient

=> Do it incrementally!
Incremental Processing

Dataset

Computation

SQL query

Result

Update

Delta

Cheaper

Recomputation

Expensive!

Incremental View Maintenance (IVM) in DBMS (Oracle, DB2, PostgreSQL, ...)

Incremental Processing

SQL query

Result

Update

Delta

Cheaper

Recomputation

Expensive!
LINVIEW

Incremental evaluation of (iterative) linear algebra programs

- APL-style programs
  - For instance: MATLAB, R, Octave
  - Matrix operations (+/-, *, \(A^T\), \(A^{-1}\))
  - Basis of ML algos

- LINVIEW compiler
  - Incremental Maintenance Optimizer
  - Code Generator

- Exec over dynamic data
  - Different runtimes (Spark, Octave)

For instance:
- MATLAB, R, Octave
- Matrix operations (+/-, *, \(A^T\), \(A^{-1}\))
- Basis of ML algos
Example: Matrix Powers $A^4$

\[
A = \begin{bmatrix}
0 & -1 & 5 \\
3 & 4 & 1 \\
-2 & 1 & -3
\end{bmatrix}
\]

\[
\Delta A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & -2 \\
0 & 0 & 0
\end{bmatrix}
\]

\[
A + \Delta A = \begin{bmatrix}
0 & -1 & 5 \\
3 & 4 & 1 \\
-2 & 1 & -3
\end{bmatrix}
\]

Incremental Program:

\[
B = AA \\
C = BB
\]

\[
O(n^3) \\
O(n^2) \\
O(n^3)
\]

\[
C = \begin{bmatrix}
35 & -47 & 224 \\
154 & 254 & 64 \\
-87 & 51 & -96
\end{bmatrix}
\]

\[
\Delta C = \begin{bmatrix}
22 & 4 & 0 \\
-42 & -54 & -120 \\
-6 & -12 & 16
\end{bmatrix}
\]

\[
C + \Delta C = \begin{bmatrix}
57 & -43 & 224 \\
112 & 200 & -56 \\
-93 & 39 & -80
\end{bmatrix}
\]
IVM of Linear Algebra

Original Program (Expensive)

\[
\text{ON UPDATE A BY } \Delta A: \\
A &= A + \Delta A \\
B &= A A \\
C &= B B
\]

\(O(n^3)\)

Incremental Program (Cheap)

\[
\text{ON UPDATE A BY } \Delta A: \\
\Delta B &= A(\Delta A) + (\Delta A)A + (\Delta A)(\Delta A) \\
\Delta C &= B(\Delta B) + (\Delta B)B + (\Delta B)(\Delta B) \\
A &= A + \Delta A \\
B &= B + \Delta B \\
C &= C + \Delta C
\]

\(O(n^2)\)

… when \(\Delta A\) is “simple”

How to

… derive delta expressions?

… evaluate delta expressions?

… represent delta expressions?
Delta Derivation

- Exploits properties of matrix operations (e.g., distributivity of matrix multiplication over addition)

Example:

\[ B[A] = AA \quad (\text{consider } B \text{ as a function of } A) \]

\[ = (A + \Delta A)(A + \Delta A) - AA \]
\[ = A(\Delta A) + (\Delta A)A + (\Delta A)(\Delta A) \]

- The Sherman–Morrison formula for maintaining \((A + \Delta A)^{-1}\)
**Delta Evaluation: The Avalanche Effect**

\[
\begin{bmatrix}
0 & -1 & 5 \\
3 & 4 & 1 \\
-2 & 1 & -3 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & -2 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
+ 
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & -2 \\
0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & -1 & 5 \\
3 & 4 & 1 \\
-2 & 1 & -3 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 & 0 & 2 \\
4 & -2 & -2 \\
0 & 0 & -2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-13 & 1 & -16 \\
10 & 14 & 16 \\
9 & 3 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 2 \\
4 & -2 & -2 \\
0 & 0 & -2 \\
\end{bmatrix}
+ 
\begin{bmatrix}
0 & 0 & 2 \\
4 & -2 & -2 \\
0 & 0 & -2 \\
\end{bmatrix}
\begin{bmatrix}
-13 & 1 & -14 \\
14 & 12 & 14 \\
9 & 3 & -2 \\
\end{bmatrix}
= 
\begin{bmatrix}
22 & 4 & 0 \\
-42 & -54 & -120 \\
-6 & -12 & 16 \\
\end{bmatrix}
\]

A single-entry change contaminates the whole output => \(\Omega(n^2)\)

Delta computation involves \(O(n^3)\) matrix multiplication

IVM loses its performance benefit over re-evaluation

How to confine the avalanche effect?
Delta Representation

- Deltas as single matrices
  - ✗ quickly escalate to full matrices, involve \( O(n^3) \) ops

- Insight: delta matrices have low ranks
  - ✓ represent as vector outer products

\[
\Delta A = \begin{bmatrix}
0 & 0 & 0 & \color{blue}{-2} \\
0 & 0 & \color{blue}{-2} & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix} = \begin{bmatrix}
0 \\
1 \\
0 \\
\end{bmatrix} \begin{bmatrix}
0 & 0 & \color{blue}{-2} \\
0 & 0 & 0 \\
\end{bmatrix} = u \, v^T
\]

- Factored representation admits efficient evaluation
Revisited: Matrix Powers $A^4$

$$\Delta A = u v^T$$

$$\Delta B = A u v^T + O(n^2) + \cdots$$

$$\Delta C = \text{a sum of 4 outer products}$$

$$\Delta A$$ is a rank-1 update

rank-s, efficient when $s << n$

Delta computation involves only $O(n^2)$ operations!
Many programs in practice converge within only a few iterations (e.g., 80.7% of pages in PageRank converge in less than 15 iterations\(^1\))

Example: \( T_i = f(A, B, T_{i-1}) = A T_{i-1} + B \)

---

**IVM of Iterative Programs**

1. Materialize
2. Derive deltas
3. Update

---

Time Complexity
(rank-1 updates, big-O notation)

<table>
<thead>
<tr>
<th></th>
<th>Re-evaluation</th>
<th>Incremental maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordinary Least Squares</td>
<td>$n^3$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Matrix Powers $A^k$</td>
<td>$n^3 \log k$</td>
<td>$n^2 k$</td>
</tr>
<tr>
<td>$T_{i+1} = AT_i + B$</td>
<td>$n^3 \log k$</td>
<td>$n^2 k$</td>
</tr>
<tr>
<td>where $T = (n \times n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{i+1} = AT_i + B$</td>
<td>$n^2 k$</td>
<td>$n^2 k$</td>
</tr>
<tr>
<td>where $T = (n \times 1)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A – dimension $(n \times n)$

IVM has lower time complexity in most cases!

But increases memory consumption ($\log k$ times, details in paper)
Experimental Setup

- Analytics: OLS, matrix powers, GD for lin. regression, ...
- Apache Spark
  - EC2 cluster: 100 workers (8 vCPUs, 13.6GB RAM, 10GbE)
- GNU Octave
  - 2 x 2.66GHz 6-Core Intel Xeon, 64GB RAM
- Randomly generated dense matrices
  - Preconditioned for numerical stability
- Stream of rank-1 updates
  - Each update affects one row of the input matrix
Matrix Powers – Scalability (nodes)

$A^{16}$ using Spark, updates to $A = (30K \times 30K)$

- Re-evaluation
- Incremental
The performance gap increases with higher dimensionality!
Ordinary Least Squares

$$\beta^* = (X^T X)^{-1} X^T Y$$

GNU Octave, updates to $$X = (n \times n)$$, $$\beta^*, Y = (n \times 1)$$

- **Re-evaluation**
- **Incremental**

<table>
<thead>
<tr>
<th>Dimension Size (n)</th>
<th>Re-evaluation</th>
<th>Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>3.6x</td>
<td></td>
</tr>
<tr>
<td>8K</td>
<td>5.2x</td>
<td></td>
</tr>
<tr>
<td>10K</td>
<td>6.3x</td>
<td></td>
</tr>
<tr>
<td>16K</td>
<td>10.6x</td>
<td></td>
</tr>
<tr>
<td>20K</td>
<td>11.5x</td>
<td></td>
</tr>
</tbody>
</table>
LINVIEW: Recap

- Incremental computation of analytical queries expressed as linear algebra programs

- Factored delta representation
  - As (sums of vector outer products)
  - Confines the avalanche effect
  - Admits efficient evaluation

- IVM has lower time complexity than re-evaluation
  - Can outperform re-evaluation by orders of magnitude

\[ \Delta A = \begin{bmatrix}
  \text{Blue} & \text{Red} & \text{Blue}
\end{bmatrix} \]

http://data.epfl.ch/linview