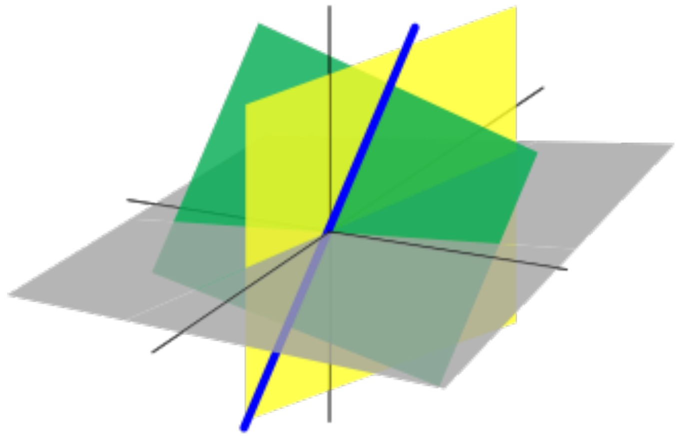


LINVIEW: Incremental View Maintenance for Complex Analytical Queries

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

SIGMOD, 24th June 2014

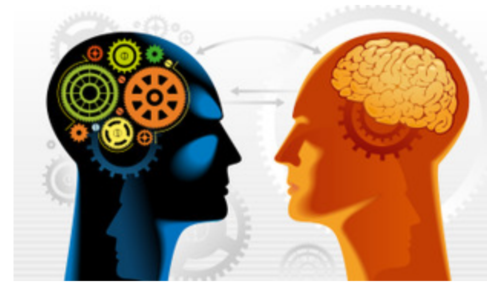
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Big Data Analytics

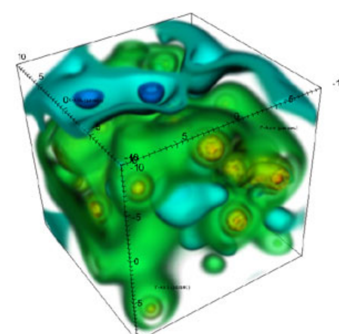
Simple (SQL) Analytics

Data Warehouses (OLAP)

Complex (non-SQL) Analytics



Machine Learning



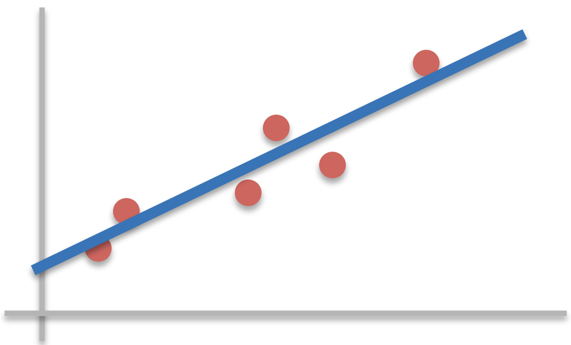
Scientific Computing

Data Mining

Complex Analytical Queries

- Often expressed as linear algebra on array data

    Example: Ordinary Least Squares



$$Y = X\beta$$

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

Re-evaluating complex queries on every (small) change is inefficient
=> Do it incrementally!

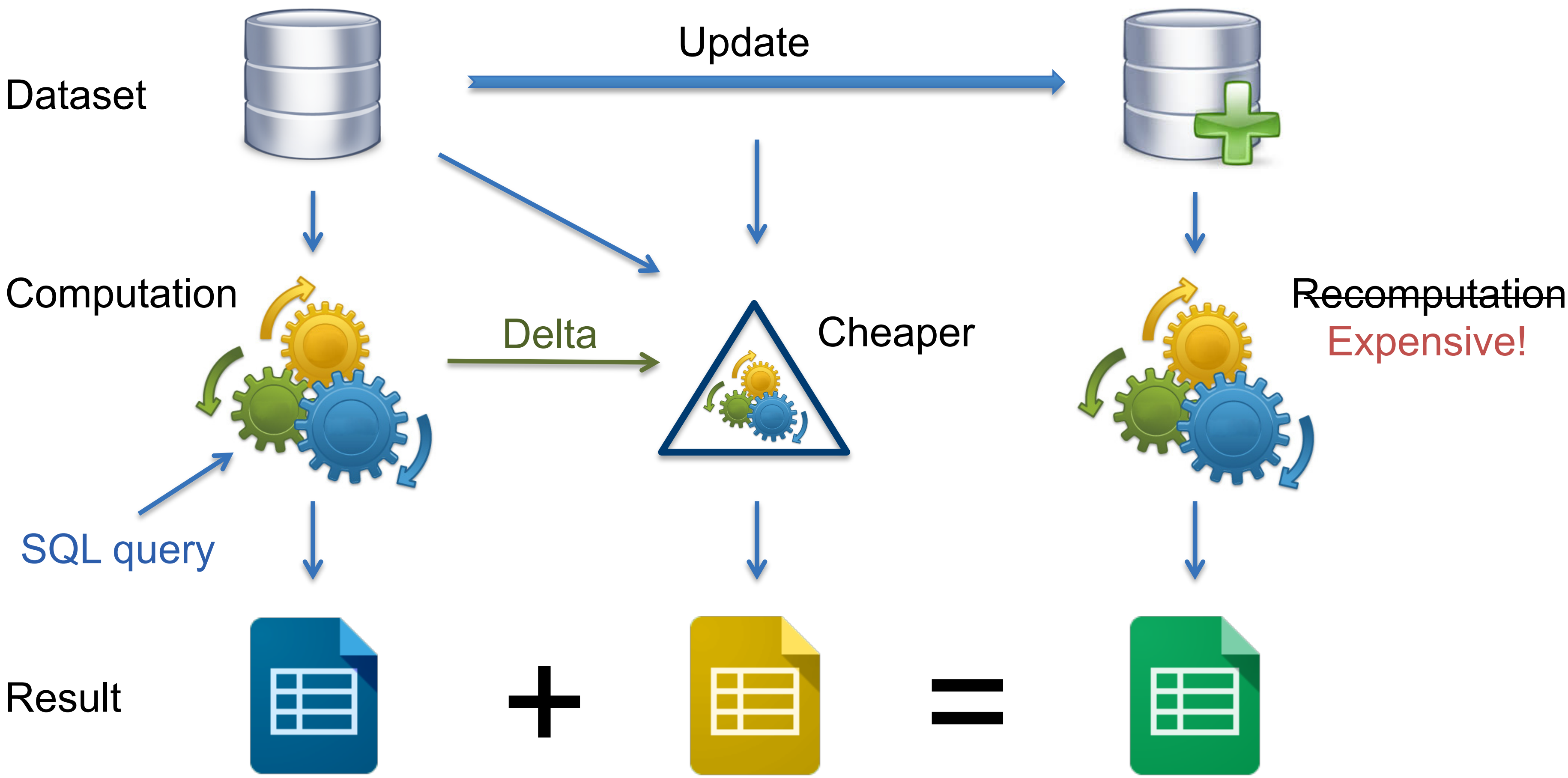
- Multidimensional arrays

    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

Incremental Processing



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

LINVIEW

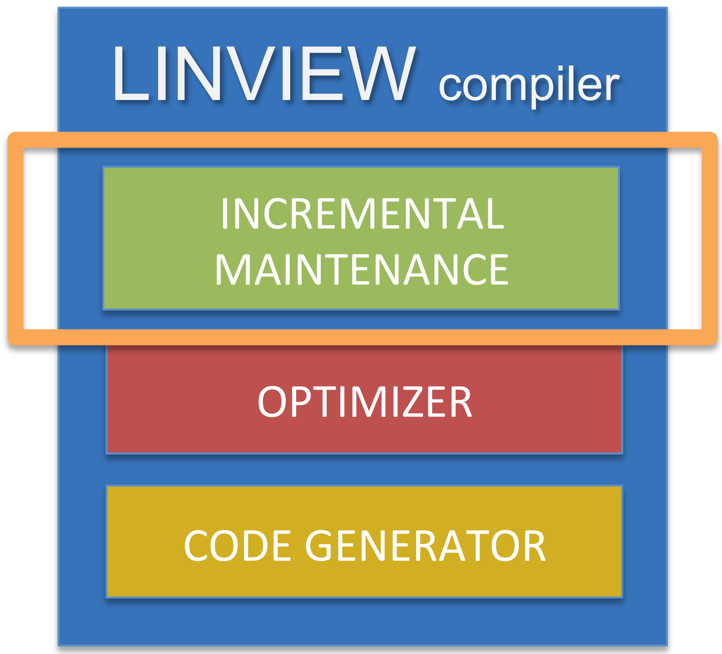
Incremental evaluation of (iterative) linear algebra programs

APL-style programs

For instance:
MATLAB, R, Octave

Matrix operations
$(+/-,\ *,\ A^T,\ A^{-1})$

Basis of ML algos



Incremental programs

Exec over dynamic data

Different runtimes
(Spark, Octave)

Example: Matrix Powers A^4

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

Delta Matrix

$B = A\,A$
$C = B\,B$
$O(n^3)$

Incremental Program
$O(n^2)$

$\Delta B = \ ...$
$\Delta C = \ ...$
$B \mathrel{+}= \Delta B$
$C \mathrel{+}= \Delta C$

$A = A + \Delta A$
$B = A\,A$
$C = B\,B$
$O(n^3)$

$$\underset{C}{\begin{bmatrix}35&-47&224\\154&254&64\\-87&51&-96\end{bmatrix}} + \underset{\Delta C}{\begin{bmatrix}22&4&0\\-42&-54&-120\\-6&-12&16\end{bmatrix}} = \underset{C + \Delta C}{\begin{bmatrix}57&-43&224\\112&200&-56\\-93&39&-80\end{bmatrix}}$$

IVM of Linear Algebra

Original Program (Expensive)

```
ON UPDATE A BY ΔA:
    A += ΔA
    B = A A
    C = B B
```

$O(n^3)$

Incremental Program (Cheap)

```
ON UPDATE A BY ΔA:
    ΔB = A(ΔA) + (ΔA)A + (ΔA)(ΔA)
    ΔC = B(ΔB) + (ΔB)B + (ΔB)(ΔB)
    A += ΔA
    B += ΔB
    C += ΔC
```

$O(n^2)$

… when ΔA 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:

  $B[A] = A\,A$ (consider B as a function of A)

  $$\begin{aligned}\Delta B[A, \Delta A] &= B[A + \Delta A] - B[A] \\ &= (A + \Delta A)(A + \Delta A) - A\,A \\ &= A(\Delta A) + (\Delta A)A + (\Delta A)(\Delta A)\end{aligned}$$

- The Sherman–Morrison formula for maintaining $(A + \Delta A)^{-1}$

Delta Evaluation: The Avalanche Effect

$$\underbrace{\begin{bmatrix} 0 & -1 & 5 \\ 3 & 4 & 1 \\ -2 & 1 & -3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix}}_{\Delta A} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix}}_{\Delta A} \underbrace{\begin{bmatrix} 0 & -1 & 5 \\ 3 & 4 & -1 \\ -2 & 1 & -3 \end{bmatrix}}_{A \,+\, \Delta A} = \underbrace{\begin{bmatrix} 0 & 0 & 2 \\ 4 & -2 & -2 \\ 0 & 0 & -2 \end{bmatrix}}_{\Delta B}$$

$$\underbrace{\begin{bmatrix} -13 & 1 & -16 \\ 10 & 14 & 16 \\ 9 & 3 & 0 \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} 0 & 0 & 2 \\ 4 & -2 & -2 \\ 0 & 0 & -2 \end{bmatrix}}_{\Delta B} + \underbrace{\begin{bmatrix} 0 & 0 & 2 \\ 4 & -2 & -2 \\ 0 & 0 & -2 \end{bmatrix}}_{\Delta B} \underbrace{\begin{bmatrix} -13 & 1 & -14 \\ 14 & 12 & 14 \\ 9 & 3 & -2 \end{bmatrix}}_{B \,+\, \Delta B} = \underbrace{\begin{bmatrix} 22 & 4 & 0 \\ -42 & -54 & -120 \\ -6 & -12 & 16 \end{bmatrix}}_{\Delta C}$$

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

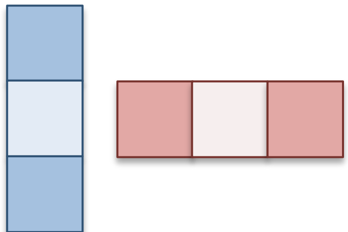
$$\Delta A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & -2 \end{bmatrix} = u\ v^T$$

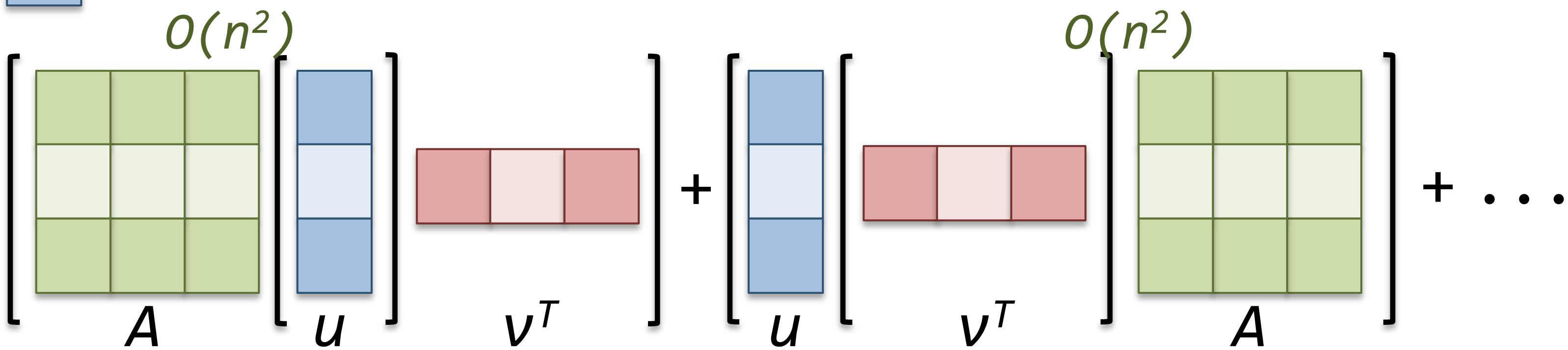
- Factored representation admits efficient evaluation

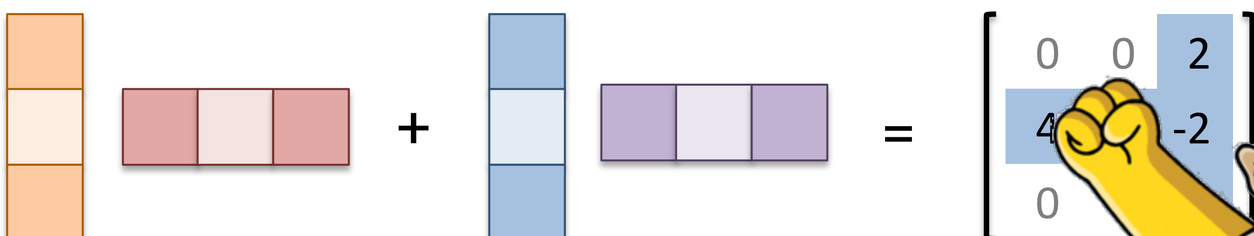
Revisited: Matrix Powers A^4

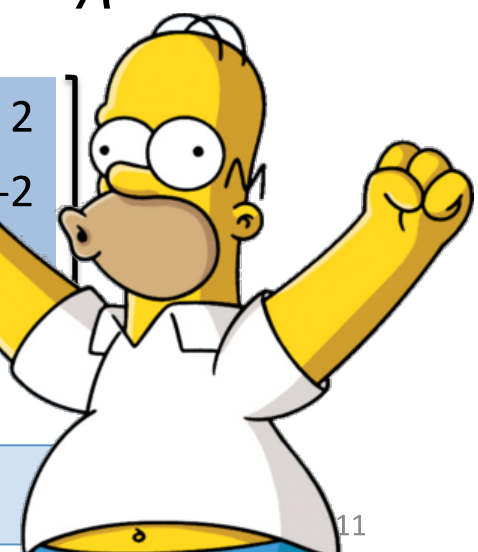
rank-s, efficient when s << n

ΔA is a ~~rank-1~~ update
(e.g., changes of one row/column)

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

$\Delta B =$ 

$=$ 

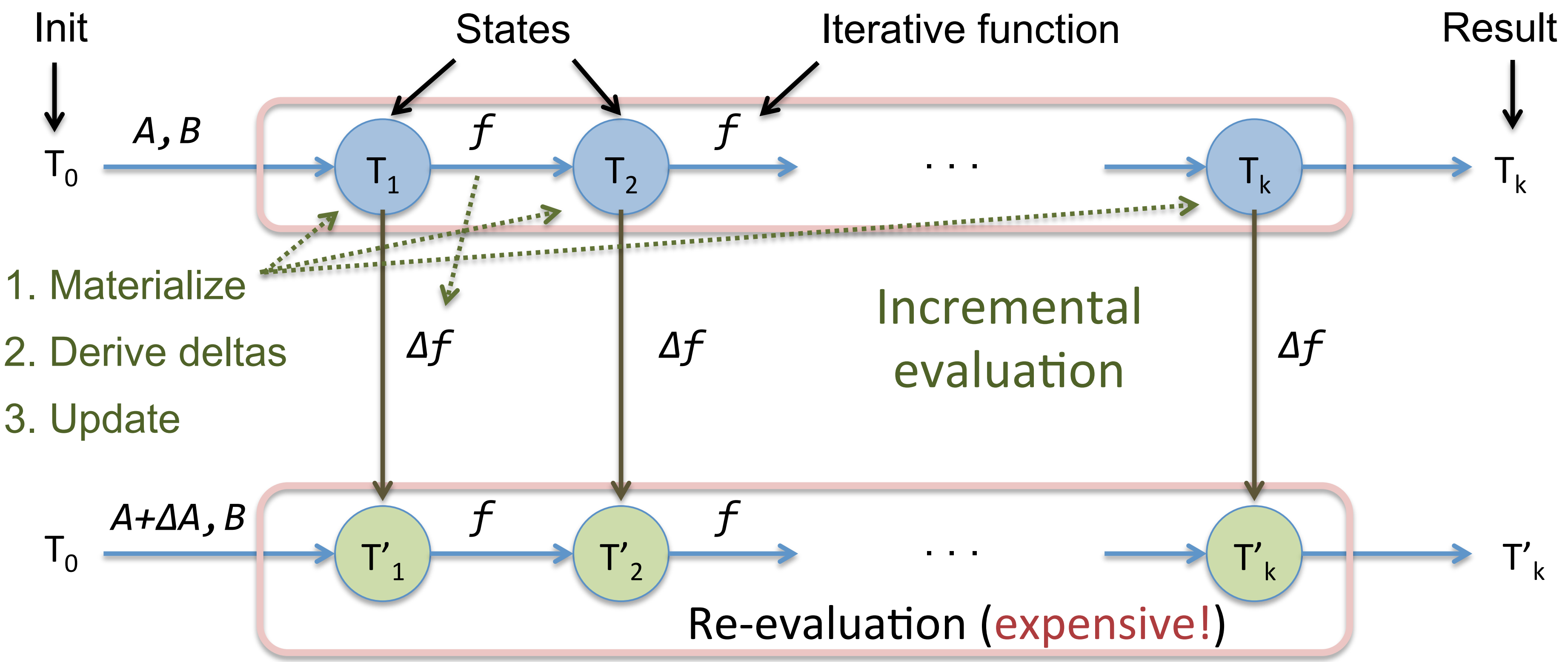


ΔC = a sum of 4 outer products

Delta computation involves only $O(n^2)$ operations!

IVM of Iterative Programs

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



[1] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of PageRank. Technical report, Stanford, 2003

Time Complexity

(rank-1 updates, big-O notation)

	Re-evaluation	Incremental maintenance
Ordinary Least Squares	n^3	n^2
Matrix Powers A^k	$n^3 \log k$	$n^2 k$
$T_{i+1} = A T_i + B$ where $T = (n \times n)$	$n^3 \log k$	$n^2 k$
$T_{i+1} = A T_i + B$ where $T = (n \times 1)$	$n^2 k$	$n^2 k$

A – dimension (n x n)

k – number of iterations

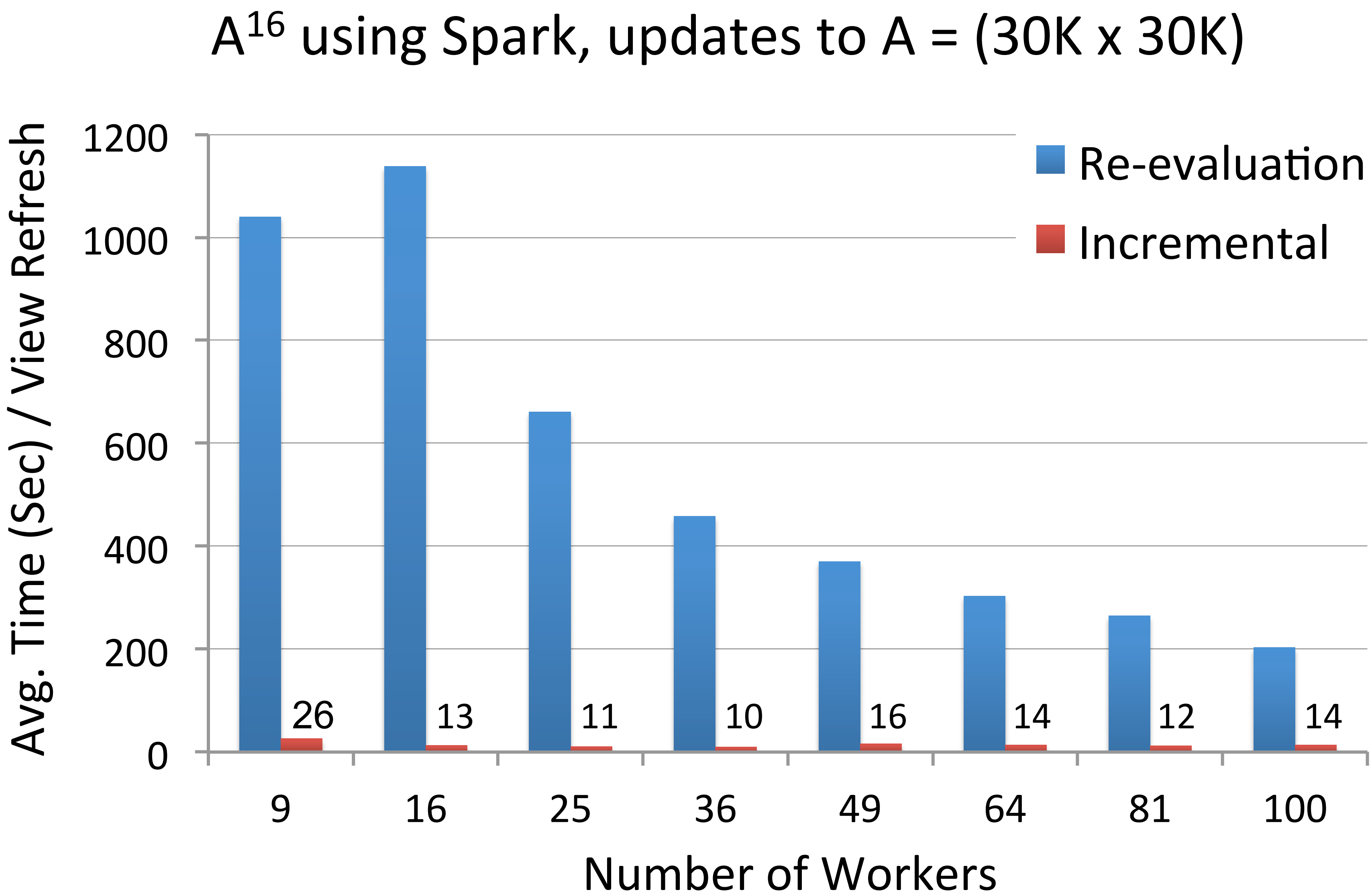
IVM has lower time complexity in most cases!

But increases memory consumption (log *k* times, details in paper)

Experimental Setup

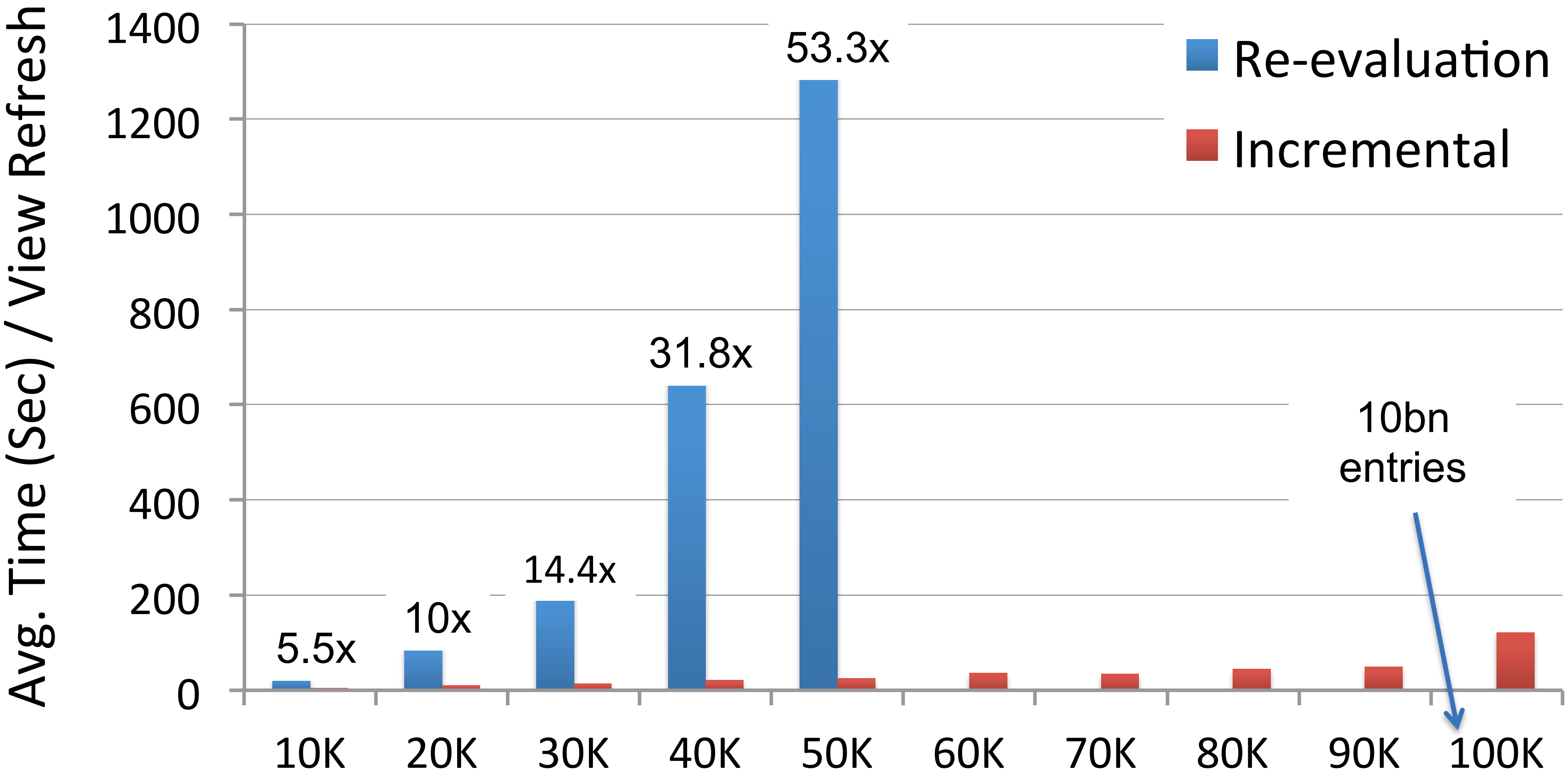
- Analytics: OLS, matrix powers, GD for lin. regression, …
- Apache Spark
  - EC2 cluster: 100 workers (8 vCPUs, 13.6GB RAM, 10GbE)
- GNU Octave
  - 2 x 2.66GHz 6-Core Intel Xeon, 64GB RAM
- Randomly generated dense matrices
  - Preconditioned for numerical stability
- Stream of rank-1 updates
  - Each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

Matrix Powers – Scalability (nodes)



Matrix Powers – Scalability (dimension)

A^{16} using 100 Spark workers, updates to A = (n x n)

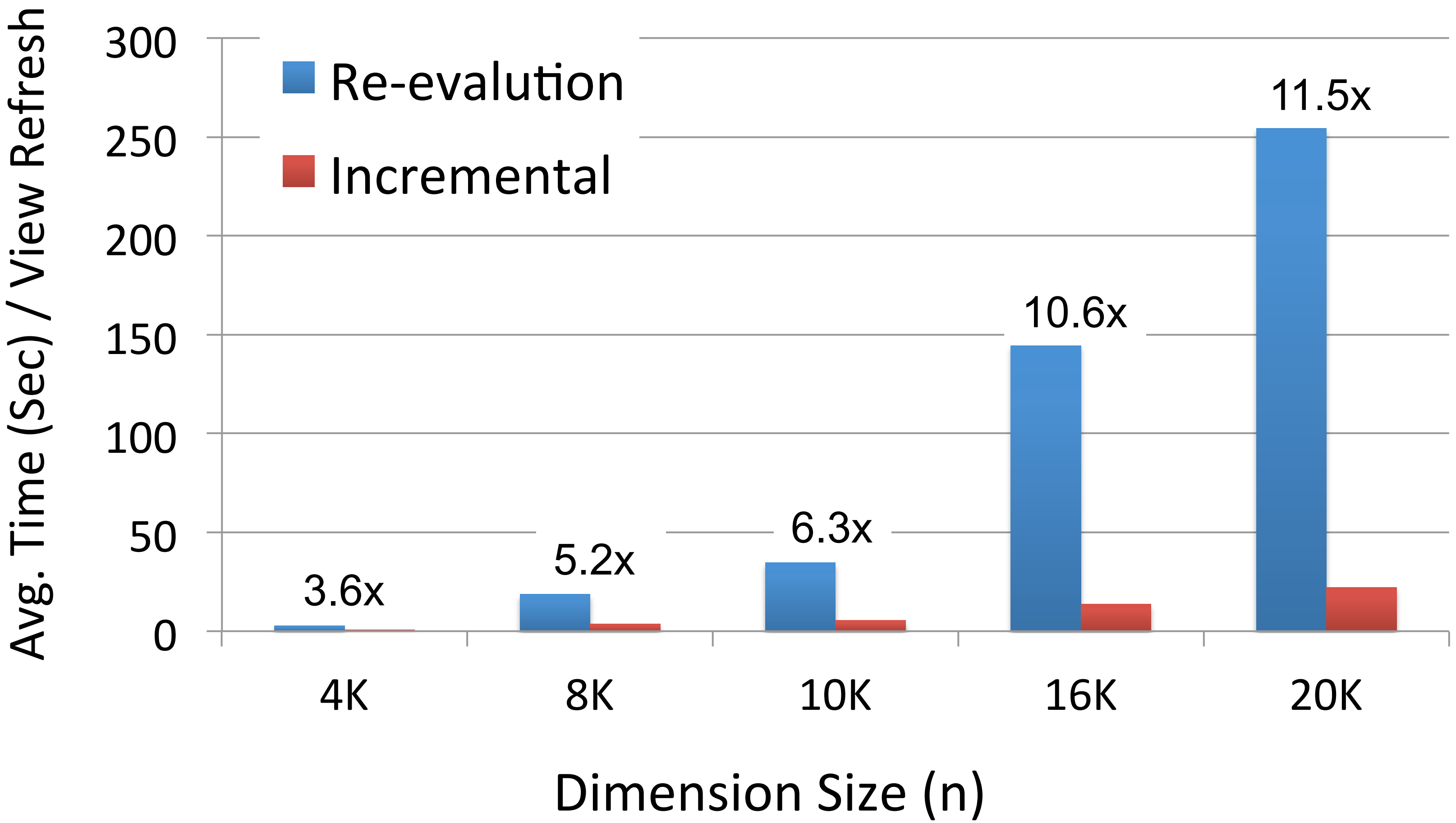


The performance gap increases with higher dimensionality!

Ordinary Least Squares

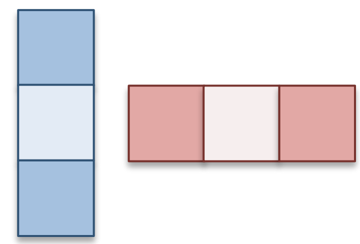
$$\beta^* = (X^T X)^{-1} X^T Y$$

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



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

$\Delta A =$ 

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

http://data.epfl.ch/linview