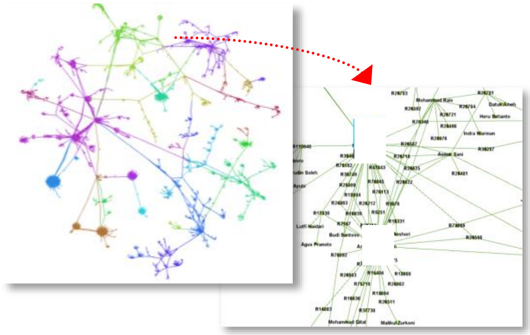
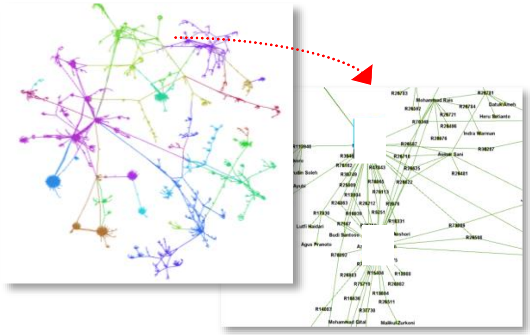
From NoSQL Accumulo to NewSQL Graphulo: Design and Utility of Graph Algorithms inside a BigTable Database

Dylan Hutchison
Jeremy Kepner
Vijay Gadepally
Bill Howe

UNIVERSITY *of* WASHINGTON
Massachusetts Institute of Technology

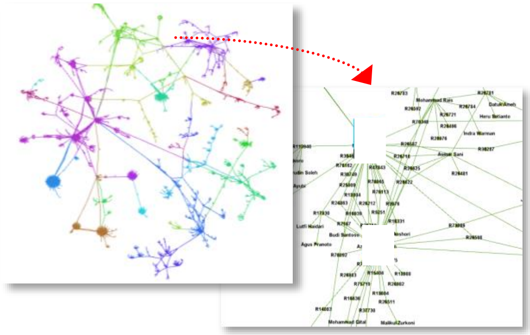


Computation ➔ Databases



Computation → Databases

Graph Algorithms → BigTable



Computation → Databases

Graph Algorithms → BigTable

GraphBLAS → accumulo™

GRAPHULO

Why push Compute into Accumulo?

> Data Locality
  - Save communication
> Reuse infrastructure
  - One less system to adopt and maintain
> Database features for free
  - Indexed access
  - Distributed execution

if the computation aligns with the DB's access path

1. ***How* to do matrix computation in Accumulo?**

2. **Applications**
   - **Jaccard coefficients**
   - **k-Truss subgraph**

3. ***When* is this a good idea?**
   - **Spoiler: Compare <u>Memory</u> and <u>I/O</u>**

Adjacency Matrix Schema

Key					Value
Row	Column			Timestamp	
	Family	Qualifier	Visibility		

- (Row, Column Qualifier, Value)
  = (v_1, v_2, weight)
  [Transpose: (v_2, v_1, weight)]

$$\begin{array}{cc} & \begin{array}{ccc} 1 & 10 & \cdots \end{array} \\ \begin{array}{c} 1 \\ 10 \\ \vdots \end{array} & \begin{bmatrix} 141 & 12 & \cdots \\ 18 & & \vdots \\ \vdots & \cdots & \ddots \end{bmatrix} \end{array}$$

Adjacency Table

```
row :colq ->val

1 :1 []   -> 141
1 :10 []   -> 12
1 :101 [] ->  9
1 :105 [] ->  3
1 :11 []   ->  9
1 :110 [] ->  3
1 :111 [] ->  3
1 :113 [] -> 12
10 :1 []   -> 18
10 :109 [] -> 2
```

Other schemas supported:
- **Graph as Incidence Matrix**
- **Graph as Single Table with degrees**

Adjacency Matrix Schema

Key					Value
Row	Column			Timestamp	
	Family	Qualifier	Visibility		

- (Row, Column Qualifier, Value) = (v_1, v_2, weight) [Transpose: (v_2, v_1, weight)]
- Degree table: store vertex degrees separately

Degree Table

```
1 :in [] ->   1084
1 :out [] -> 1027
10 :in [] ->  118
10 :out [] ->  94
100 :in [] ->   8
100 :out [] -> 10
```

$$\begin{array}{cc} & \begin{array}{ccc} 1 & 10 & \cdots \end{array} \\ \begin{array}{c} 1 \\ 10 \\ \vdots \end{array} & \begin{bmatrix} 141 & 12 & \cdots \\ 18 & & \vdots \\ \vdots & \cdots & \ddots \end{bmatrix} \end{array}$$

Adjacency Table

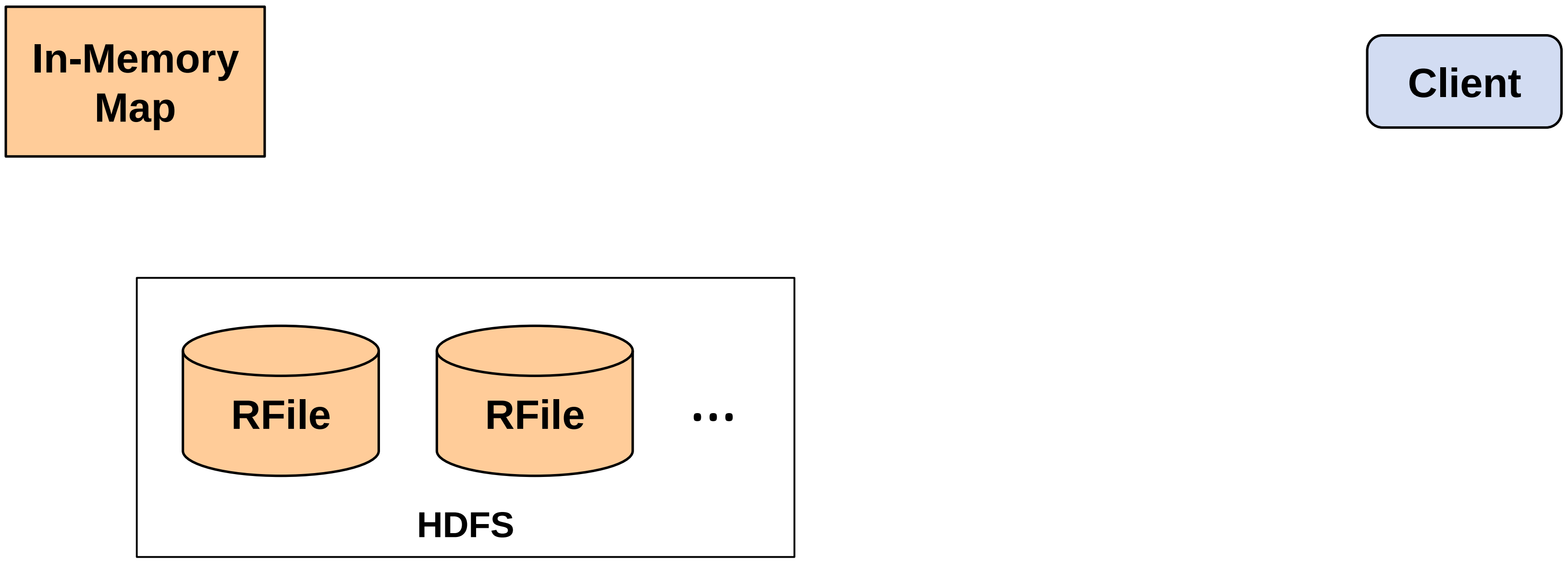
```
row :colq ->val

1 :1 []   -> 141
1 :10 []   -> 12
1 :101 [] ->  9
1 :105 [] ->  3
1 :11 []   ->  9
1 :110 [] ->  3
1 :111 [] ->  3
1 :113 [] -> 12
10 :1 []   -> 18
10 :109 [] -> 2
```

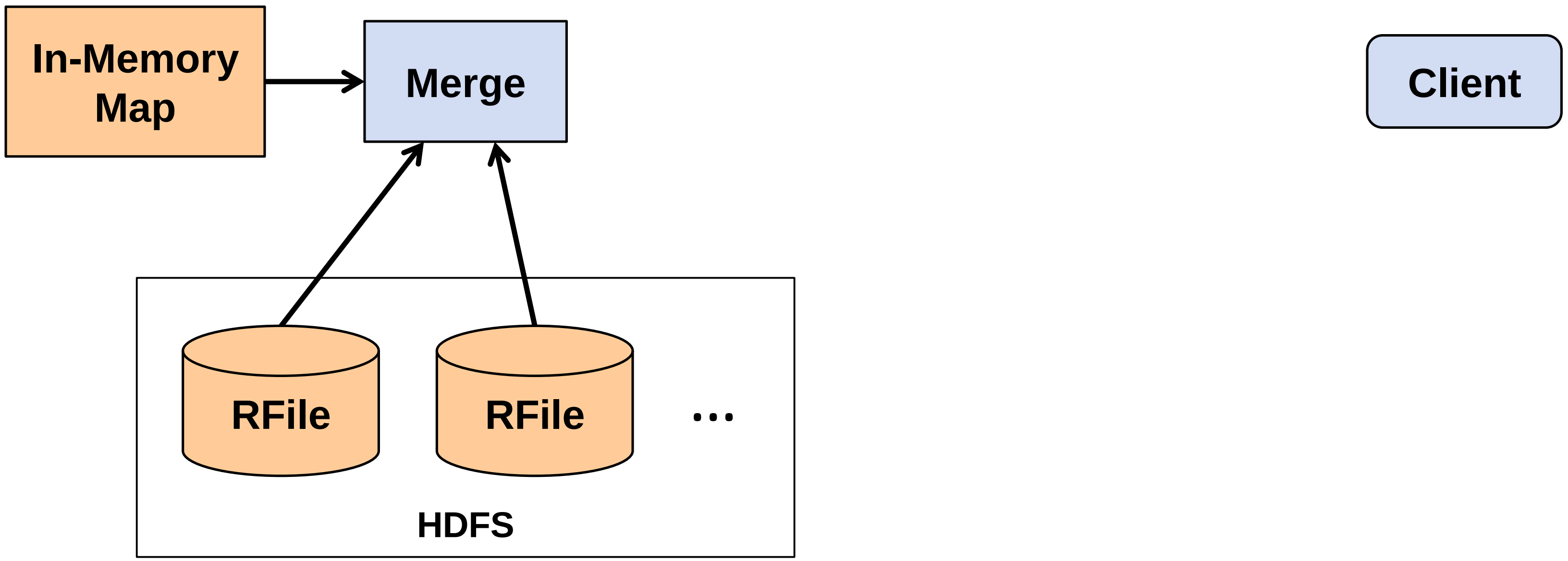
Other schemas supported:

- Graph as Incidence Matrix
- Graph as Single Table with degrees

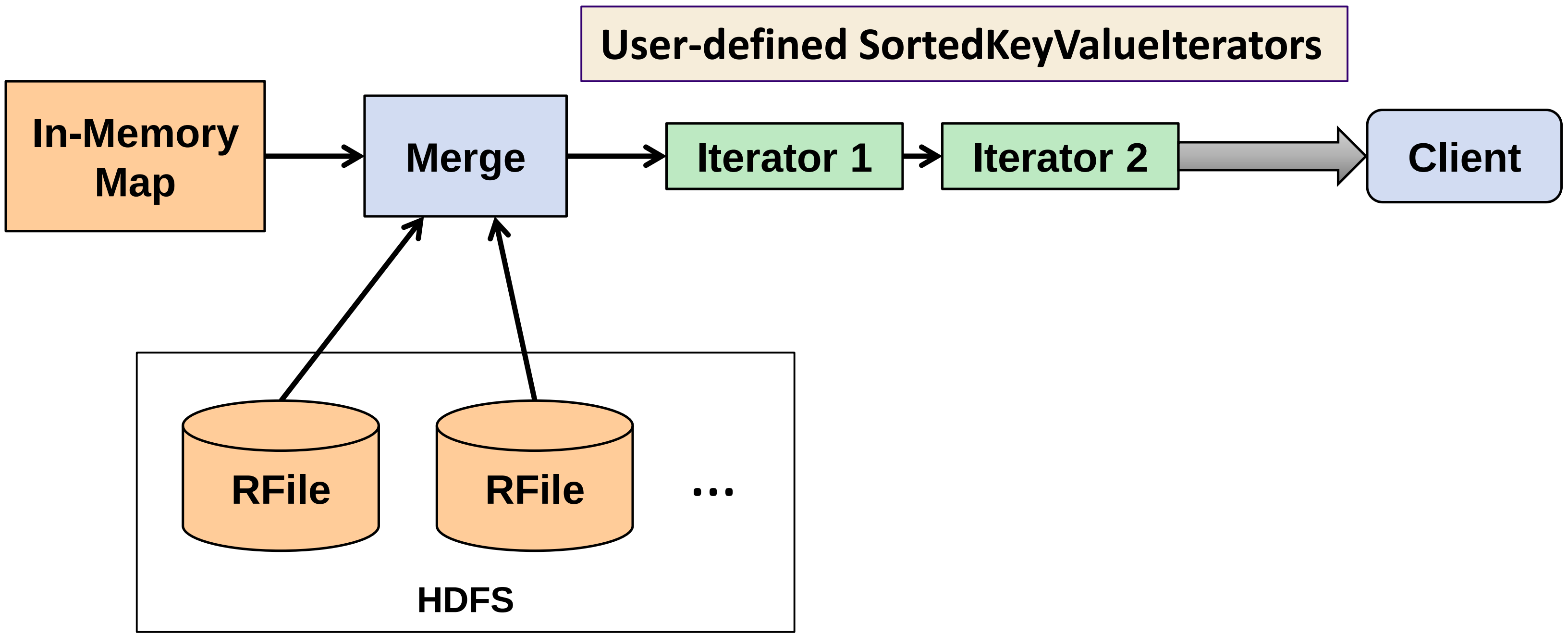
Accumulo Scan Iterator Pipeline



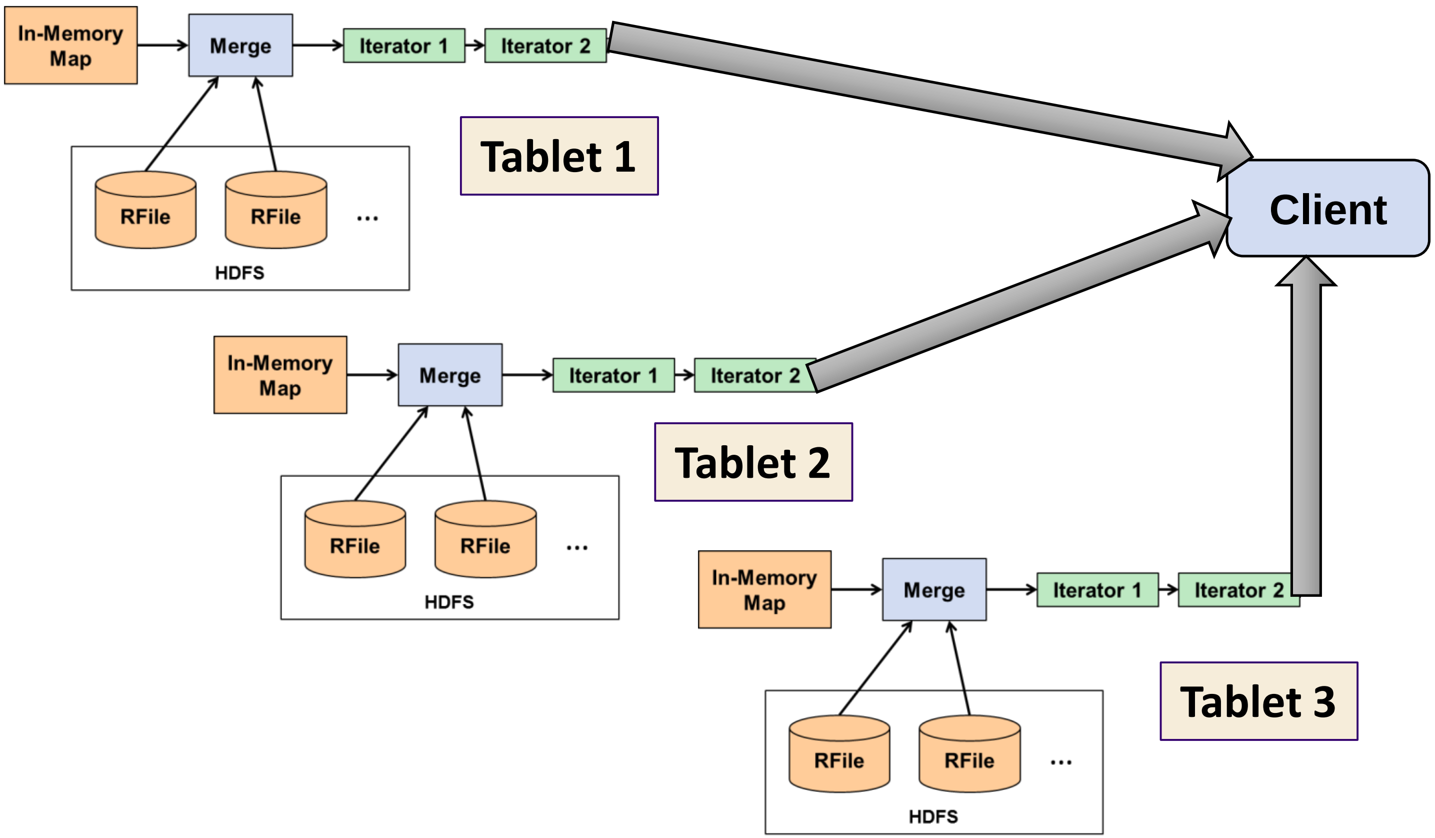
Accumulo Scan Iterator Pipeline



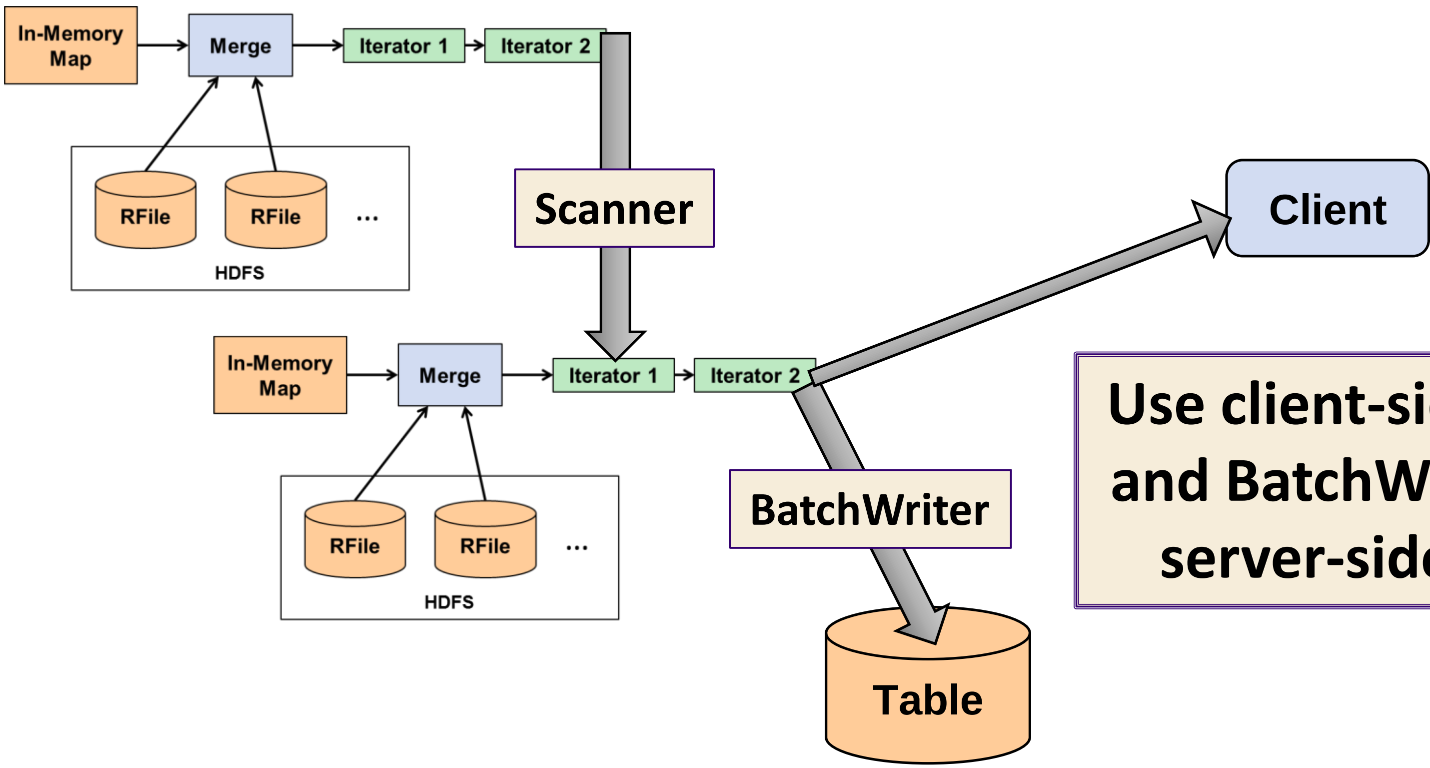
Accumulo Scan Iterator Pipeline



Accumulo BatchScan Iterator Pipeline

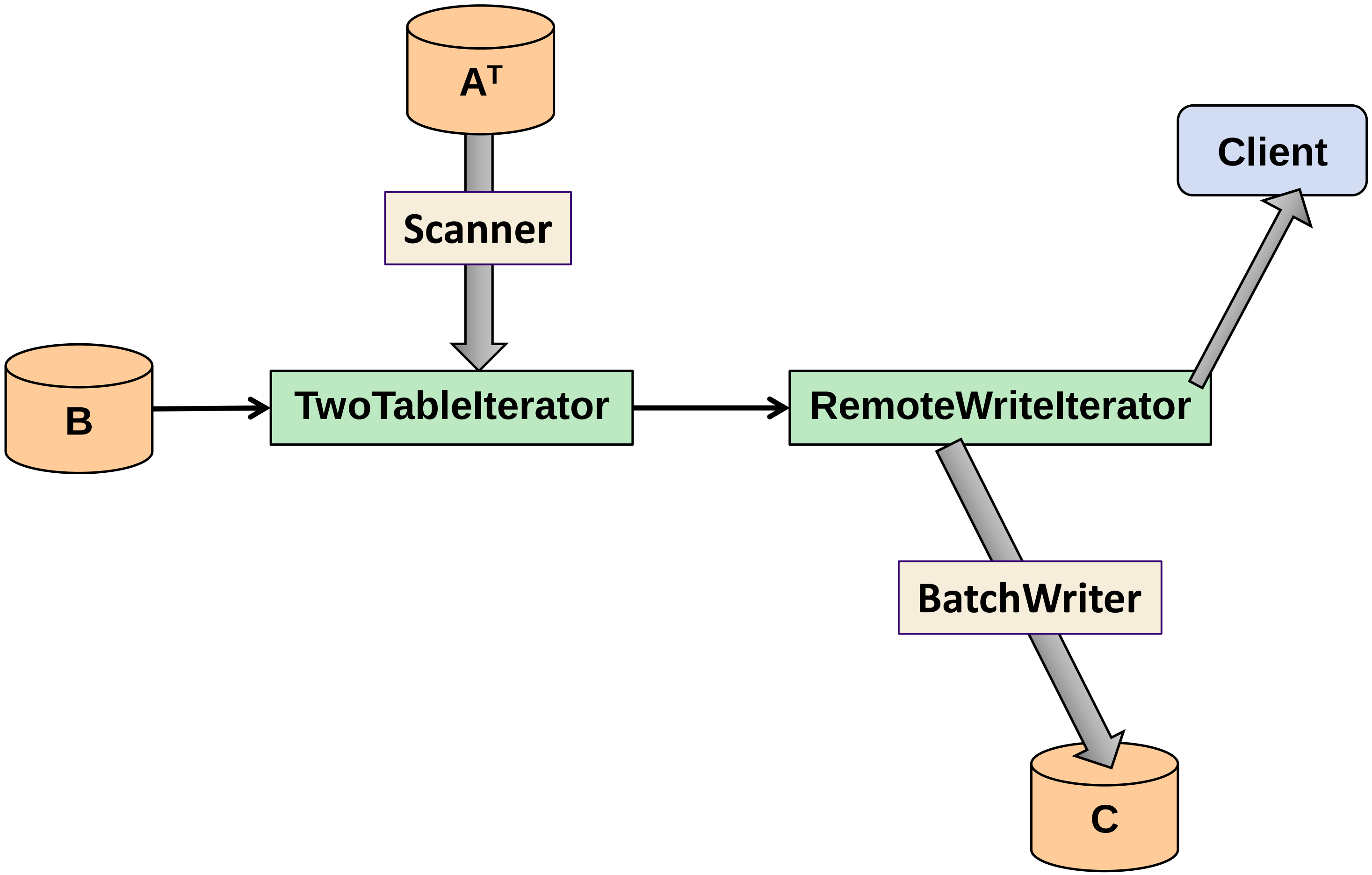


Graphulo addition to Iterator Pipeline

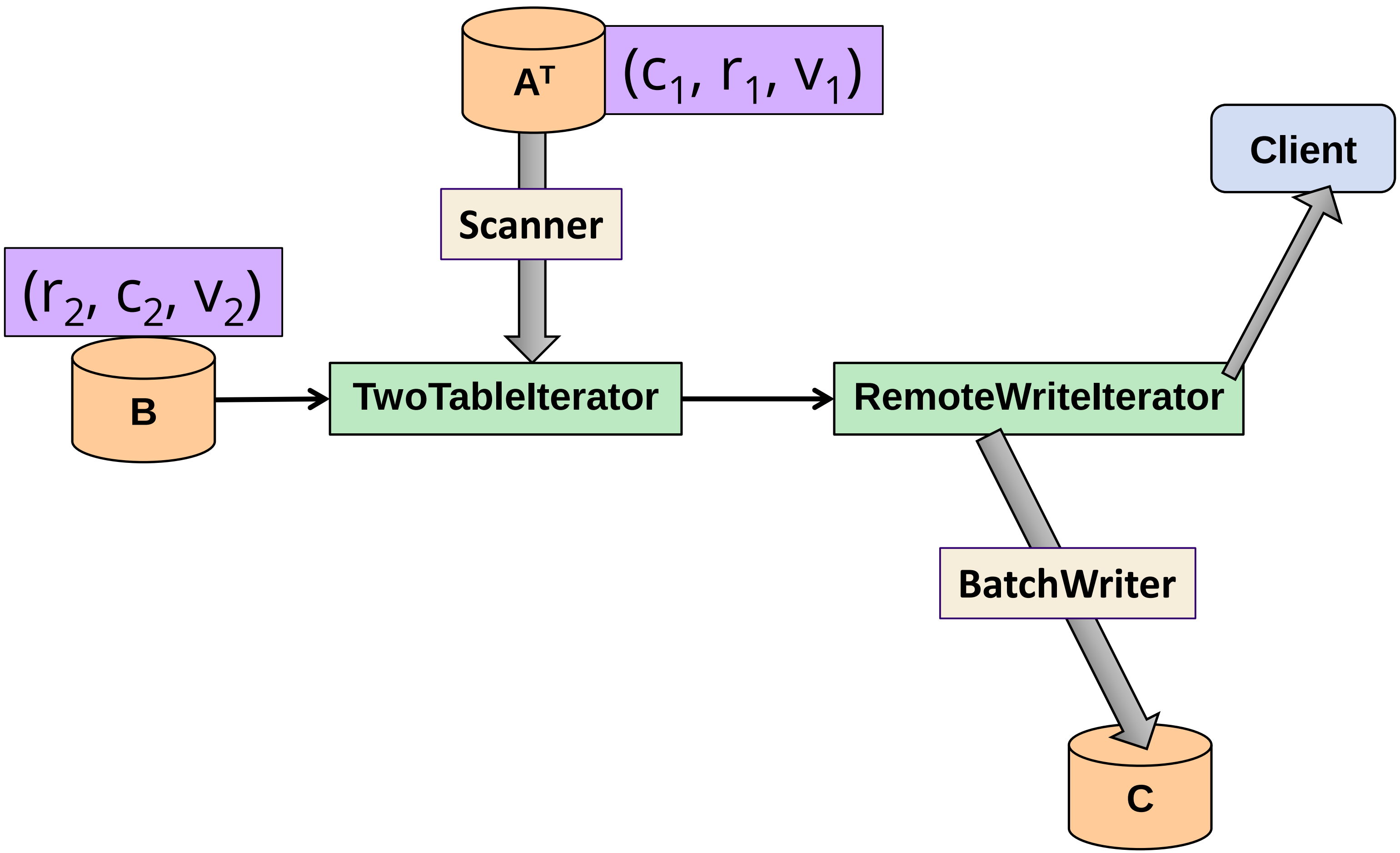


Use client-side Scanners and BatchWriters inside server-side iterators

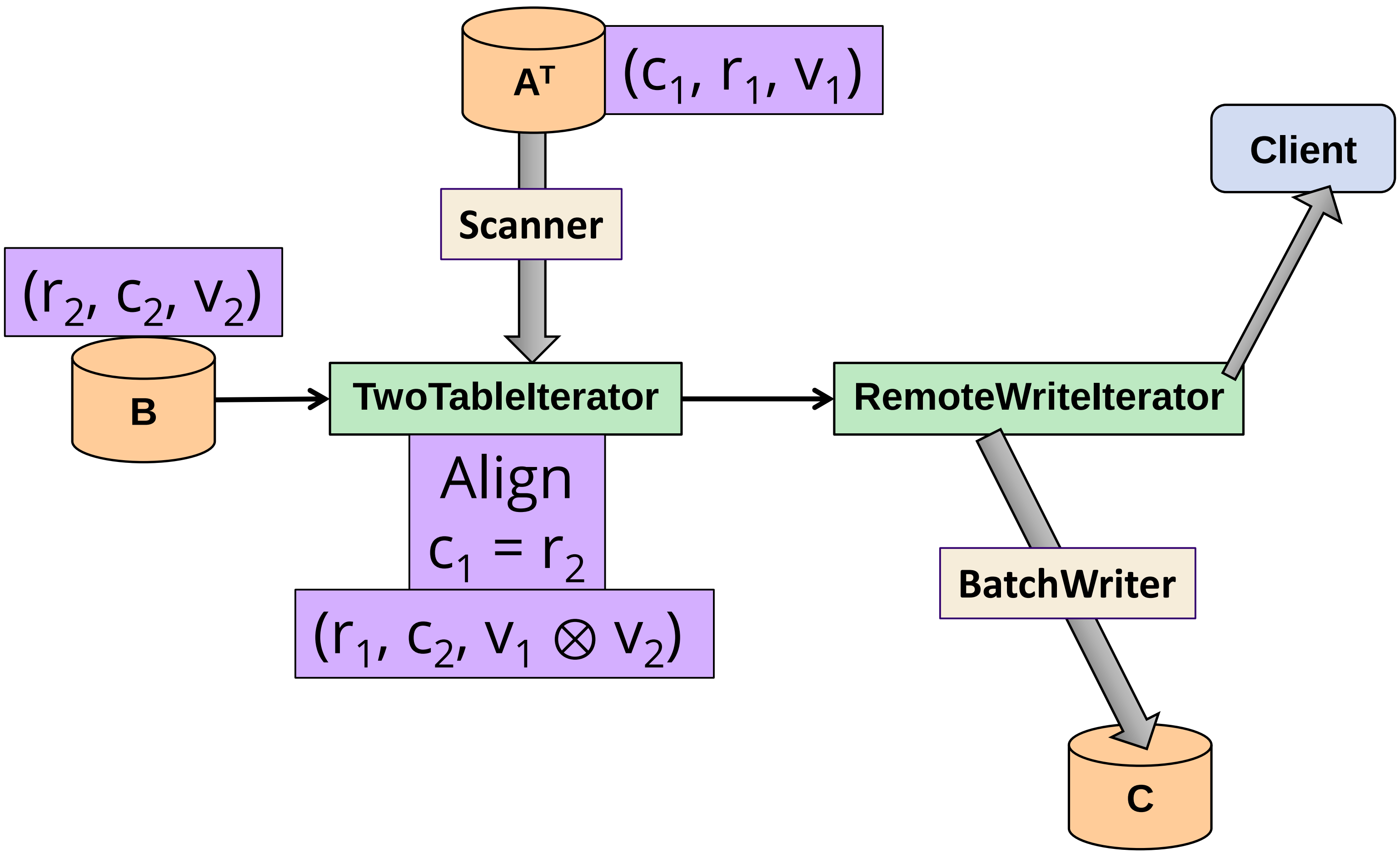
Graphulo MxM: $A^T (\oplus . \otimes) B$



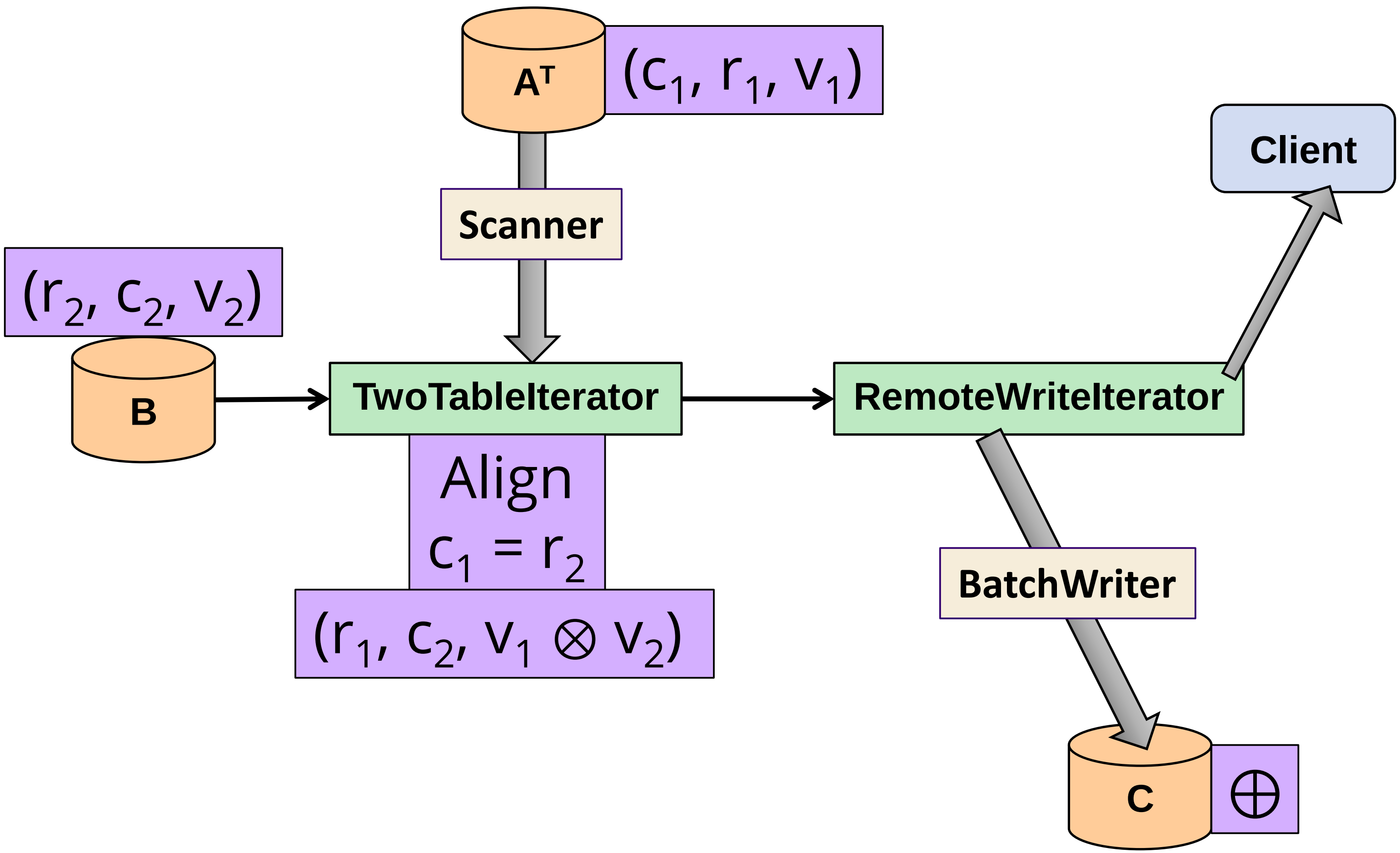
Graphulo MxM: $A^T(\oplus.\otimes)B$



Graphulo MxM: $A^T (\oplus . \otimes) B$



Graphulo MxM: $A^T (\oplus.\otimes)$ B



Graphulo Client Functions

```
long TableMult(String ATtable, String Btable, String Ctable)


long SpEWiseX(String Atable, String Btable, String Ctable)
long SpEWiseSum(String Atable, String Btable, String Ctable)
…
```

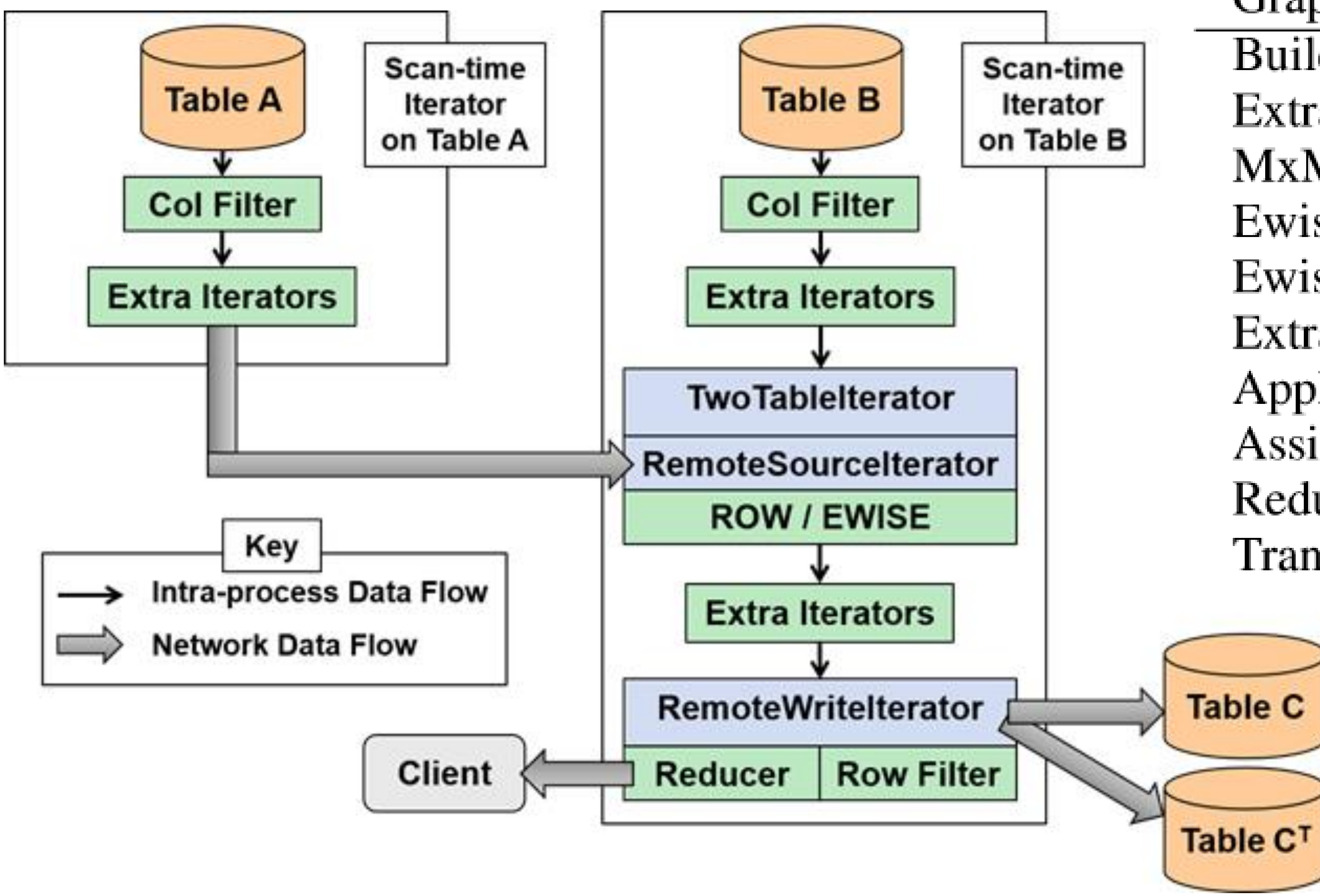
Simple API abstracts the iterator pipeline

Graphulo Client Functions

```
long TwoTable(
    String ATtable, String Btable, String Ctable, String CTtable,
    int BScanIteratorPriority, TwoTableIterator.DOTMODE dotmode,
    Map<String, String> optsTT, IteratorSetting plusOp,
    String rowFilter, String colFilterAT, String colFilterB,
    boolean emitNoMatchA, boolean emitNoMatchB,
    List<IteratorSetting> iteratorsBeforeA,
    List<IteratorSetting> iteratorsBeforeB,
    List<IteratorSetting> iteratorsAfterTwoTable,
    Reducer reducer, Map<String, String> reducerOpts,
    int numEntriesCheckpoint, Authorizations ATauthorizations,
    Authorizations Bauthorizations, int batchWriterThreads
)
```

Full control
when you need it

Graphulo's TwoTable Pipeline

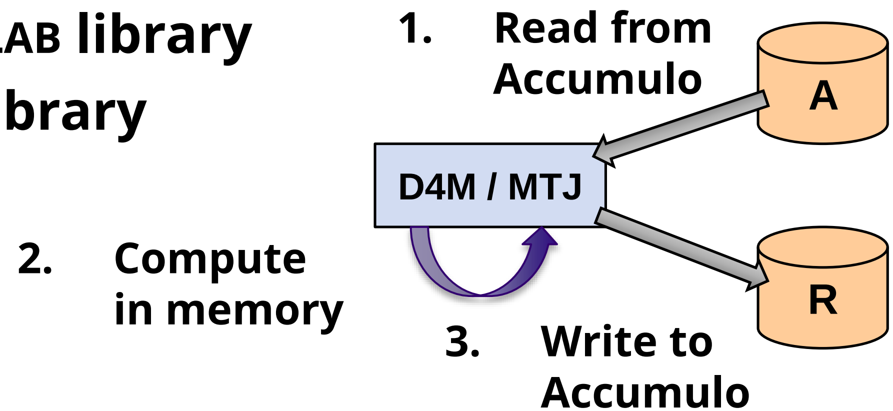


GraphBLAS Kernel
BuildMatrix $(\oplus)$
ExtracTuples
MxM $(\oplus, \otimes)$
EwiseMult $(\otimes)$
EwiseAdd $(\oplus)$
Extract
Apply (f)
Assign
Reduce $(\oplus)$
Transpose

Applications and Performance

Applications and Performance

- Algorithms: Jaccard and k-Truss
- Comparison to main-memory alternatives
  - Sparse: D4M MATLAB library
  - Dense: MTJ Java library



- Node: 16GB RAM, 8 i7 cores, Accumulo 1.8.0
- Data: Unpermuted power law graph generator
  - 2^{SCALE} nodes, 16 edges/node
- Accumulo Threads: 1-2 tablets

Alg 1: Jaccard Coefficients

- **Neighborhood overlap of two vertices**
- **Vertex similarity measure**

$$J_{ij} = \frac{| N(v_i) \bigcap N(v_j) |}{| N(v_i) \bigcup N(v_j) |}$$

```
long Jaccard(String Aorig, String ADeg, String Rfinal)
```

Graphulo: Linear algebra graph kernels for NoSQL databases, Gadepally et al, IEEE IPDPSW 2015

Alg 1: Jaccard Coefficients

Input: Unweighted, undirected adjacency matrix $\mathbf{A}$

Output: Upper triangle of Jaccard coefficients $\mathbf{J}$

```
d = sum(A)                                // pre-computed in degree table
U = triu(A, 1)                            // strict upper triangle filter
J = triu(UU + UUᵀ + UᵀU, 1)               // fused MxM
foreach nonzero entry J_ij ∈ J do
    J_ij = J_ij / (d_i + d_j − J_ij)      // stateful Apply on J
end
```

$\mathbf{d} = \mathrm{sum}(\mathbf{A})$ *// pre-computed in degree table*

$\mathbf{U} = \mathrm{triu}(\mathbf{A}, 1)$ *// strict upper triangle filter*

$\mathbf{J} = \mathrm{triu}(\mathbf{U}\mathbf{U} + \mathbf{U}\mathbf{U}^\intercal + \mathbf{U}^\intercal\mathbf{U}, 1)$ *// fused MxM*

foreach *nonzero entry* $\mathbf{J}_{ij} \in \mathbf{J}$ **do**

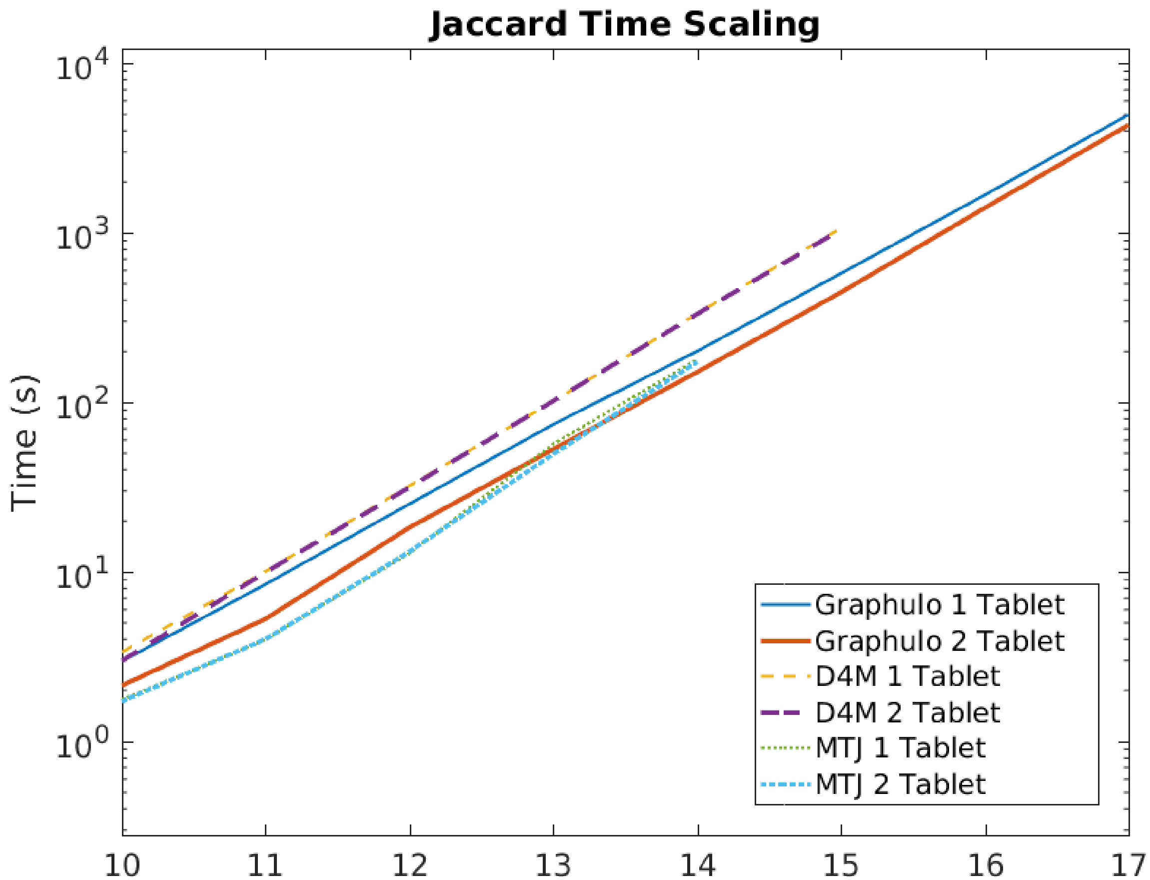
  $\mathbf{J}_{ij} = \mathbf{J}_{ij} / (\mathbf{d}_i + \mathbf{d}_j - \mathbf{J}_{ij})$ *// stateful Apply on* $\mathbf{J}$

end

Algorithm 1: Jaccard

Fusion:
A single Graphulo
TwoTable pass!

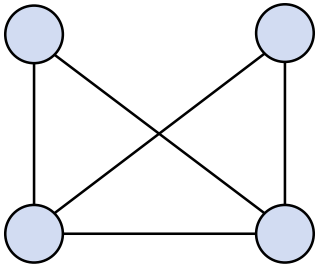
Jaccard Performance



Problem size = 2^ SCALE nodes

Alg 2: k–Truss Subgraph

- A graph is a k-Truss if each edge is part of at least k-2 triangles
  - May be the empty graph
- Indicates a graph's "core community"



Example 3-truss

```
long kTrussAdj(String Aorig, String Rfinal, int k)
```

Alg 2: k-Truss Subgraph

Input: Unweighted, undirected adjacency matrix $\mathbf{A}_0$, integer k

Output: Adjacency matrix of k-truss subgraph $\mathbf{A}$

$z' = \infty, \mathbf{A} = \mathbf{A}_0$ *// table clone*

repeat

  $z = z'$

  $\mathbf{B} = \mathbf{A}$ *// table clone*

  $\mathbf{B} = \mathbf{B} + 2\mathbf{A}\mathbf{A}$ *// MxM with $a \otimes b = 2$ if $a, b \neq 0$*

  $\mathbf{B}(\mathbf{B} \,\%\, 2 == 0) = 0$ *// filter on $\mathbf{B}$*

  $\mathbf{B}((\mathbf{B} - 1)/2 < k - 2) = 0$ *// filter on $\mathbf{B}$*

  $\mathbf{A} = |\mathbf{B}|_0$ *// Apply on $\mathbf{B}$; switch $\mathbf{A} \leftrightarrow \mathbf{B}$*

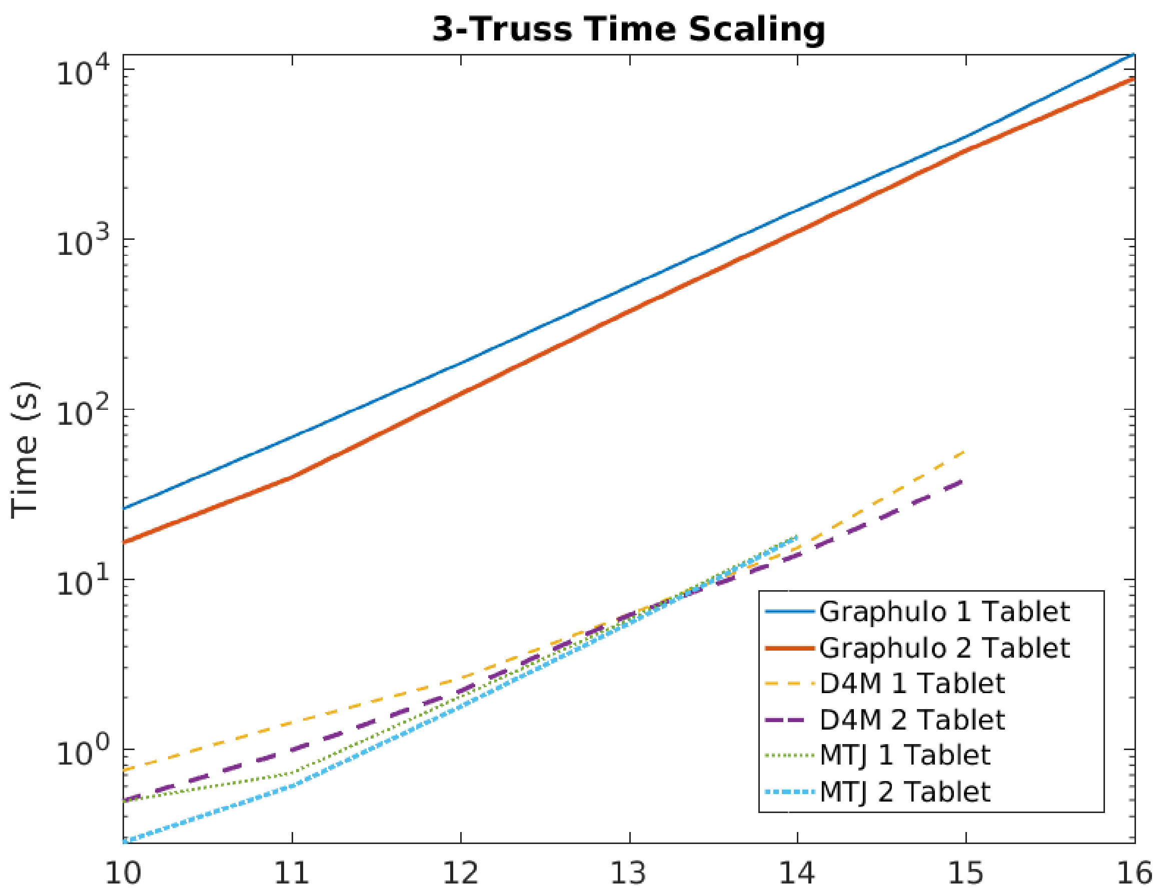
  $z' = \mathrm{nnz}(\mathbf{A})$ *// Reduce, gathering nnz at client*

until $z == z'$ *// client controls iteration*

Algorithm 2: kTruss

Iterations of Graphulo TwoTables

3-Truss Performance



Problem size = 2^ SCALE nodes

Performance Diff

SCALE	nnz($\mathbf{A}$)	nnz(Jaccard($\mathbf{A}$))	Partial Products	Graphulo Overhead	Graphulo 1 Tablet	Graphulo 2 Tablets	D4M 1 Tablet	D4M 2 Tablets	MTJ 1 Tablet	MTJ 2 Tablets
10	2.10×10^4	2.15×10^5	1.01×10^6	4.7x	2.97	2.14	3.36	2.99	1.76	1.72
11	4.52×10^4	7.07×10^5	3.10×10^6	4.4x	8.46	5.29	1.01×10^1	9.96	3.99	4.01
12	9.67×10^4	2.18×10^6	9.29×10^6	4.3x	2.52×10^1	1.83×10^1	3.22×10^1	3.16×10^1	1.29×10^1	1.32×10^1
13	2.04×10^5	6.75×10^6	2.71×10^7	4.0x	7.42×10^1	5.30×10^1	1.02×10^2	1.02×10^2	5.68×10^1	4.96×10^1
14	4.26×10^5	2.02×10^7	7.77×10^7	3.8x	2.01×10^2	1.51×10^2	3.34×10^2	3.33×10^2	1.79×10^2	1.73×10^2
15	8.83×10^5	6.07×10^7	2.22×10^8	3.7x	5.77×10^2	4.46×10^2	1.07×10^3	1.05×10^3		
16	1.82×10^6	1.77×10^8	6.20×10^8	3.5x	1.68×10^3	1.41×10^3				
17	3.73×10^6	5.16×10^8	1.72×10^9	3.3x	4.99×10^3	4.34×10^3				

TABLE II: Jaccard experiment statistics. Graphulo is competitive and better scales due to low overhead.

SCALE	nnz($\mathbf{A}$)	nnz(3Truss($\mathbf{A}$))	Partial Products	Graphulo Overhead	Graphulo 1 Tablet	Graphulo 2 Tablets	D4M 1 Tablet	D4M 2 Tablets	MTJ 1 Tablet	MTJ 2 Tablets
10	2.10×10^4	2.03×10^4	5.94×10^6	293.3x	2.57×10^1	1.63×10^1	0.74	0.49	0.49	0.28
11	4.52×10^4	4.35×10^4	1.22×10^7	280.7x	6.78×10^1	3.93×10^1	1.42	0.98	0.72	0.60
12	9.67×10^4	9.20×10^4	5.45×10^7	592.7x	1.84×10^2	1.21×10^2	2.58	2.18	2.02	1.76
13	2.04×10^5	1.93×10^5	1.59×10^8	825.5x	5.22×10^2	3.72×10^2	6.16	6.09	5.74	5.44
14	4.26×10^5	3.99×10^5	4.55×10^8	1140.6x	1.47×10^3	1.10×10^3	1.52×10^1	1.38×10^1	1.79×10^1	1.75×10^1
15	8.83×10^5	8.20×10^5	1.30×10^9	1582.5x	3.97×10^3	3.29×10^3	5.65×10^1	3.82×10^1		
16	1.82×10^6	1.67×10^6	3.62×10^9	2167.0x	1.22×10^4	8.77×10^3				

TABLE III: 3Truss experiment statistics. D4M and MTJ execute faster, assuming sufficient memory, due to high overhead. Graphulo overhead is [illegible] e listed in seconds.

Graphulo Overhead: How many more entries Graphulo writes to Accumulo than the main-memory systems

Performance Diff

(Overhead is due to Graphulo writing all partial products whereas main-memory systems pre-sum)

SCALE	nnz(**A**)	nnz(Jaccard(**A**))	Partial Products	Graphulo Overhead	Graphulo 1 Tablet	Graphulo 2 Tablets	D4M 1 Tablet	D4M 2 Tablets	MTJ 1 Tablet	MTJ 2 Tablets
10	2.10×10^4	2.15×10^5	1.01×10^6	4.7x	2.97	[illegible]	[illegible]	[illegible]	[illegible]	1.72
11	4.52×10^4	7.07×10^5	3.10×10^6	4.4x	8.46	[illegible]	[illegible]	[illegible]	[illegible]	4.01
12	9.67×10^4	2.18×10^6	9.29×10^6	4.3x	2.52×10^1	[illegible]	[illegible]	[illegible]	[illegible]	1.32×10^1
13	2.04×10^5	6.75×10^6	2.71×10^7	4.0x	7.42×10^1	[illegible]	[illegible]	[illegible]	[illegible]	4.96×10^1
14	4.26×10^5	2.02×10^7	7.77×10^7	3.8x	2.01×10^2	[illegible]	[illegible]	[illegible]	[illegible]	1.73×10^2
15	8.83×10^5	6.07×10^7	2.22×10^8	3.7x	5.77×10^2	[illegible]	[illegible]	[illegible]		
16	1.82×10^6	1.77×10^8	6.20×10^8	3.5x	1.68×10^3	1.41×10^3				
17	3.73×10^6	5.16×10^8	1.72×10^9	3.3x	4.99×10^3	4.34×10^3				

Comparable I/O; savings in communication outweighs I/O overhead

TABLE II: Jaccard experiment statistics. Graphulo is competitive and better scales due to low overhead.

SCALE	nnz(**A**)	nnz(3Truss(**A**))	Partial Products	Graphulo Overhead	Graphulo 1 Tablet	Graphulo 2 Tablets	D4M 1 Tablet	D4M 2 Tablets	MTJ 1 Tablet	MTJ 2 Tablets
10	2.10×10^4	2.03×10^4	5.94×10^6	293.3x	2.57×10^1	1.63×10^1	0.74	0.49	0.49	0.28
11	4.52×10^4	4.35×10^4	1.22×10^7	280.7x	6.78×10^1	3.93×10^1	1.42	0.98	0.72	0.60
12	9.67×10^4	9.20×10^4	5.45×10^7	592.7x	1.84×10^2	1.21×10^2	2.58	2.18	2.02	1.76
13	2.04×10^5	1.93×10^5	1.59×10^8	825.5x	5.22×10^2	3.72×10^2	6.16	6.09	5.74	5.44
14	4.26×10^5	3.99×10^5	4.55×10^8	1140.6x	1.47×10^3	1.10×10^3	1.52×10^1	1.38×10^1	1.79×10^1	1.75×10^1
15	8.83×10^5	8.20×10^5	1.30×10^9	1582.5x	3.97×10^3	3.29×10^3	5.65×10^1	3.82×10^1		
16	1.82×10^6	1.67×10^6	3.62×10^9	2167.0x	1.22×10^4	8.77×10^3				

TABLE III: 3Truss experiment statistics. D4M and MTJ execute faster, assuming sufficient memory, due to high overhead. Graphulo overhead is [illegible] are listed in seconds.

Graphulo Overhead: How many more entries Graphulo writes to Accumulo than the main-memory systems

Performance Diff

(Overhead is due to Graphulo writing all partial products whereas main-memory systems pre-sum)

SCALE	nnz(**A**)	nnz(Jaccard(**A**))	Partial Products	Graphulo Overhead	Graphulo 1 Tablet	Graphulo 2 Tablets	D4M 1 Tablet	D4M 2 Tablets	MTJ 1 Tablet	MTJ 2 Tablets
10	2.10×10^4	2.15×10^5	1.01×10^6	4.7x	2.97	[illegible]	[illegible]	[illegible]	[illegible]	1.72
11	4.52×10^4	7.07×10^5	3.10×10^6	4.4x	8.46	[illegible]	[illegible]	[illegible]	[illegible]	4.01
12	9.67×10^4	2.18×10^6	9.29×10^6	4.3x	2.52×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.32×10^1
13	2.04×10^5	6.75×10^6	2.71×10^7	4.0x	7.42×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4.96×10^1
14	4.26×10^5	2.02×10^7	7.77×10^7	3.8x	2.01×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.73×10^2
15	8.83×10^5	6.07×10^7	2.22×10^8	3.7x	5.77×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
16	1.82×10^6	1.77×10^8	6.20×10^8	3.5x	1.68×10^3	1.41×10^3				
17	3.73×10^6	5.16×10^8	1.72×10^9	3.3x	4.99×10^3	4.34×10^3				

Comparable I/O;
savings in communication outweighs I/O overhead

TABLE II: Jaccard experiment statistics. Graphulo is competitive and better scales due to low overhead.

SCALE	nnz(**A**)	nnz(3Truss(**A**))	Partial Products	Graphulo Overhead	Graphulo 1 Tablet	Graphulo 2 Tablets	D4M 1 Tablet	D4M 2 Tablets	MTJ 1 Tablet	MTJ 2 Tablets
10	2.10×10^4	2.03×10^4	5.94×10^6	293.3x	2.57×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.28
11	4.52×10^4	4.35×10^4	1.22×10^7	280.7x	6.78×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.60
12	9.67×10^4	9.20×10^4	5.45×10^7	592.7x	1.84×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.76
13	2.04×10^5	1.93×10^5	1.59×10^8	825.5x	5.22×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5.44
14	4.26×10^5	3.99×10^5	4.55×10^8	1140.6x	1.47×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.75×10^1
15	8.83×10^5	8.20×10^5	1.30×10^9	1582.5x	3.97×10[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
16	1.82×10^6	1.67×10^6	3.62×10^9	2167.0x	1.22×10^4	8.77×10^3				

Vastly different I/O;
main-memory systems cache temporary tables whereas Graphulo writes all to Accumulo

TABLE III: 3Truss experiment statistics. D4M and MTJ execute faster, assuming sufficient memory, due to high overhead.
Graphulo overhead is [illegible] are listed in seconds.

Graphulo Overhead: How many more entries Graphulo writes to Accumulo than the main-memory systems

GRAPHULO

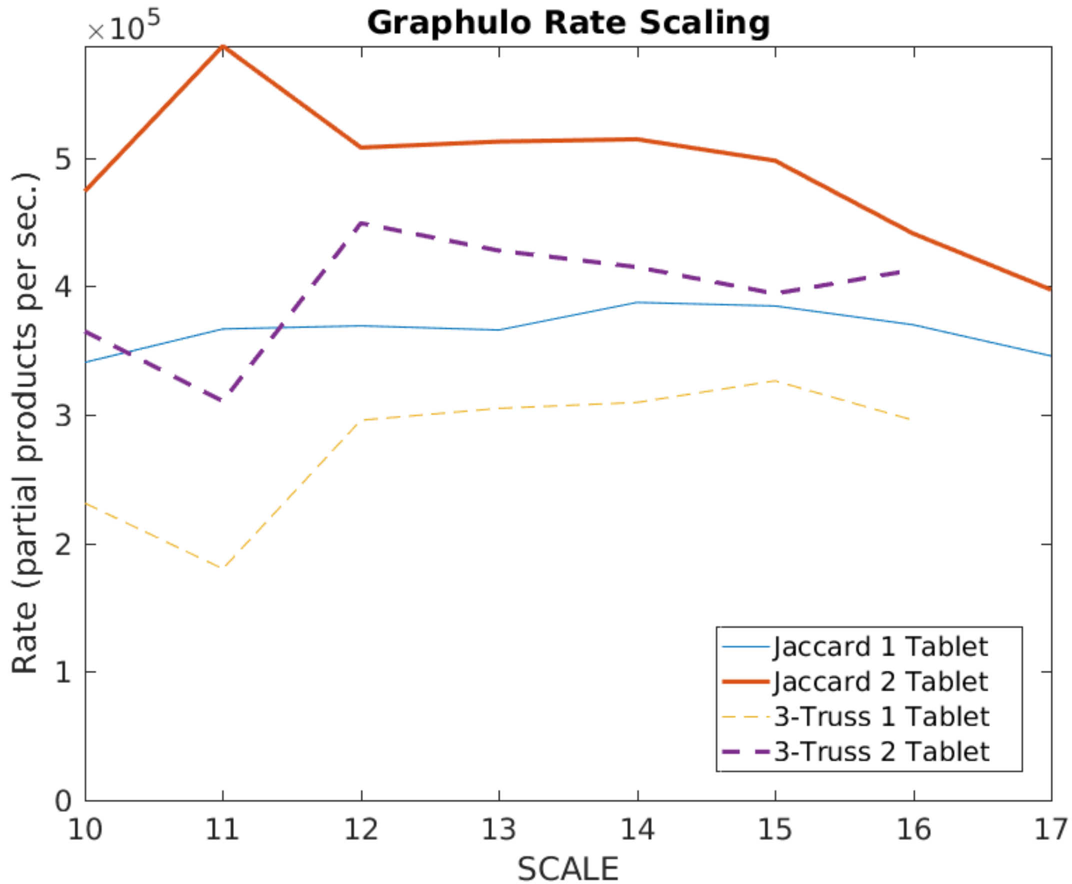
http://graphulo.mit.edu

> *How* to do matrix math in Accumulo?
  - The TwoTable iterator pipeline
> Jaccard & k-Truss
> *When* to do matrix math in Accumulo?
  - Memory requirements
  - Compare in-database I/O vs. alternatives
> Future Work: *Multi-Node*,
  expand to *Relational Algebra*,
  use an *Optimizer* to choose the best plan

Contact: Dylan Hutchison
dhutchis@cs.washington.edu

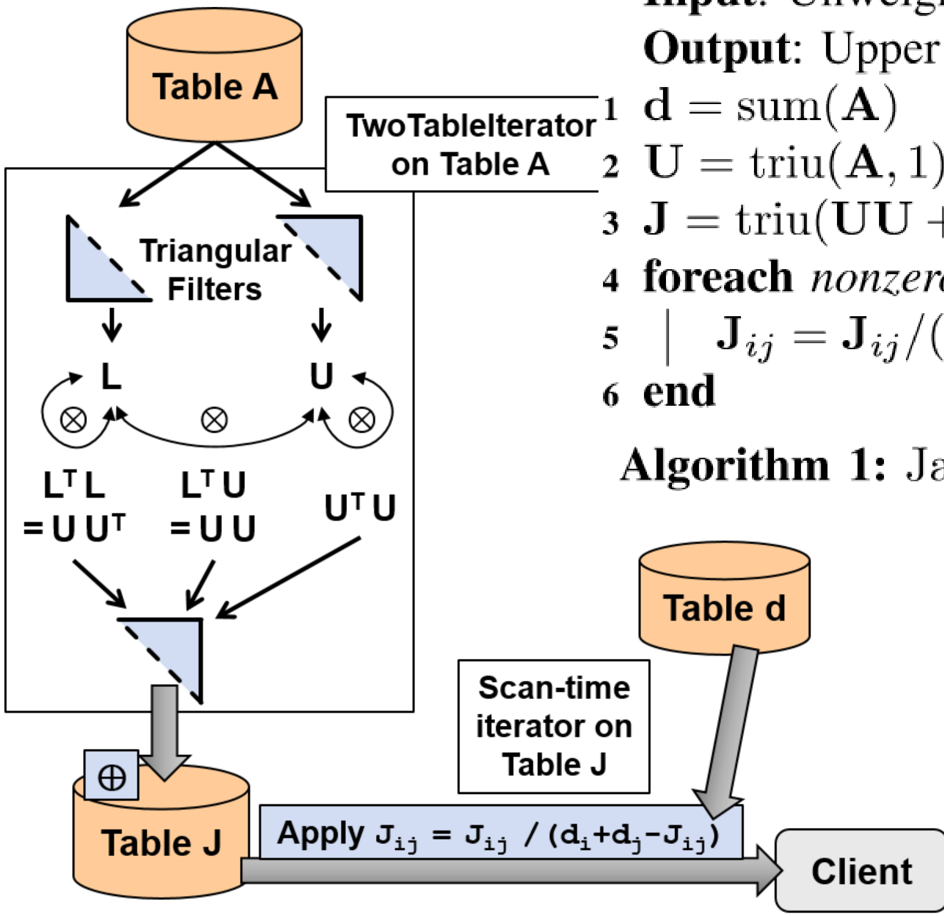
Backup

Overall Graphulo Performance



Problem size = 2^ nodes

Alg 1: Jaccard Coefficients



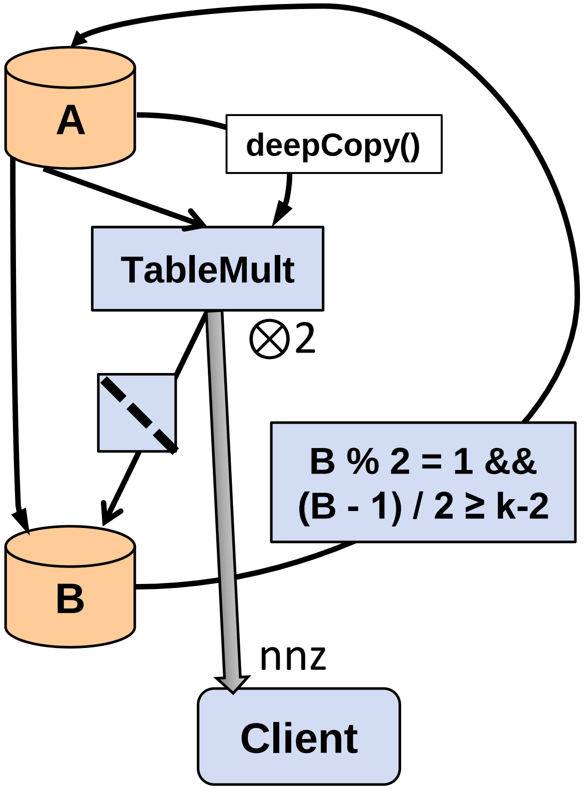
Input: Unweighted, undirected adjacency matrix $\mathbf{A}$
Output: Upper triangle of Jaccard coefficients $\mathbf{J}$

$\mathbf{d} = \mathrm{sum}(\mathbf{A})$ *// pre-computed in degree table*
$\mathbf{U} = \mathrm{triu}(\mathbf{A}, 1)$ *// strict upper triangle filter*
$\mathbf{J} = \mathrm{triu}(\mathbf{UU} + \mathbf{UU}^\intercal + \mathbf{U}^\intercal\mathbf{U}, 1)$ *// fused MxM*
foreach *nonzero entry* $\mathbf{J}_{ij} \in \mathbf{J}$ **do**
  $\mathbf{J}_{ij} = \mathbf{J}_{ij} / (\mathbf{d}_i + \mathbf{d}_j - \mathbf{J}_{ij})$ *// stateful Apply on* $\mathbf{J}$
end

Algorithm 1: Jaccard

Fusion:
A single Graphulo TwoTable pass!

Alg 2: k-Truss Subgraph



Input: Unweighted, undirected adjacency matrix $\mathbf{A}_0$, integer k

Output: Adjacency matrix of k-truss subgraph $\mathbf{A}$

$z' = \infty, \mathbf{A} = \mathbf{A}_0$ *// table clone*

repeat

  $z = z'$

  $\mathbf{B} = \mathbf{A}$ *// table clone*

  $\mathbf{B} = \mathbf{B} + 2\mathbf{A}\mathbf{A}$ *// MxM with $a \otimes b = 2$ if $a, b \neq 0$*

  $\mathbf{B}(\mathbf{B} \,\%\, 2 == 0) = 0$ *// filter on $\mathbf{B}$*

  $\mathbf{B}((\mathbf{B} - 1)/2 < k - 2) = 0$ *// filter on $\mathbf{B}$*

  $\mathbf{A} = |\mathbf{B}|_0$ *// Apply on $\mathbf{B}$; switch $\mathbf{A} \leftrightarrow \mathbf{B}$*

  $z' = \mathrm{nnz}(\mathbf{A})$ *// Reduce, gathering nnz at client*

until $z == z'$ *// client controls iteration*

Algorithm 2: kTruss

Iterations of Graphulo TwoTables

Background on Accumulo

Key					Value
Row	Column			Timestamp	
	Family	Qualifier	Visibility		

Best for:

- Large, de-normalized tables; no schema necessary
  - Unlimited columns; un-interpreted values; everything is a byte[]
- TBs to PBs of data; robust horizontal scaling
- Hadoop HDFS / Java ecosystem
- Cell-level visibility
- Row store by default
  - Scan over rows for O(log n) lookup & sorted order
  - Use Transpose Tables for column indexing[1]
- Iterator processing framework

[1] *D4M 2.0 Schema: A General Purpose High Performance Schema for the Accumulo Database*, Kepner et al, IEEE HPEC 2013

GraphBLAS Operations

GraphBLAS Kernel	Graphulo Implementation
BuildMatrix ($\oplus$)	Accumulo BatchWriter
ExtracTuples	Accumulo BatchScanner
MxM ($\oplus, \otimes$)	TwoTableIterator ROW mode, performing $\mathbf{A}^\intercal\mathbf{B}$
EwiseMult ($\otimes$)	TwoTableIterator EWISE mode
EwiseAdd ($\oplus$)	Similar to EwiseMult, with non-matching entries
Extract	Row and column filtering
Apply (f)	Extra Iterators
Assign	Apply with a key-transforming function
Reduce ($\oplus$)	Reducer module on RemoteWriteIterator
Transpose	Transpose option on RemoteWriteIterator