In-Database Analytics for NoSQL Key-Value Stores

Dylan Hutchison, 5 December 2016
Advisors: Bill Howe, Dan Suciu

UNIVERSITY of WASHINGTON

Tons of NoSQL Key-Value Stores!

amazon DynamoDB

Project Voldemort
A distributed database.

riak

accumulo™

levelDB

ORACLE®
BERKELEY DB

Microsoft Azure
DocumentDB

APACHE HBASE

cassandra

Who uses a K-V DB? How?

Basho Riak Use Cases

➔ Find a "natural key" for your application

DATA TYPE	KEY	VALUE
Session	User/Session ID	Session Data
Content	ID	Documents, Images, Posts, Videos, Texts, JSON/HTML, etc.
Advertising	Campaign ID	Ad Content
Logs	Date	Log File
Sensor	Date/Time	Sensor Updates
User Data	Login, Email, UUID	User Attributes

http://basho.com/products/riak-kv/

Who uses a K-V DB? How?

Timely – Time Series Event Monitoring

Row	ColumnFamily	ColumnQualifier	Value
sys.cpu.user\1447879348291	host=r001n01	instance=0,rack=r001	2.0
sys.cpu.user\1447879348291	instance=0	host=r001n01,rack=r001	2.0
sys.cpu.user\1447879348291	rack=r001	host=r001n01,instance=0	2.0

- Compound key
- Intentional order
- Multiple values per key
- Repeated values

Many tricks to encode information into K-Vs

https://nationalsecurityagency.github.io/timely/docs/#data-storage-example

Who uses a K–V DB? How?

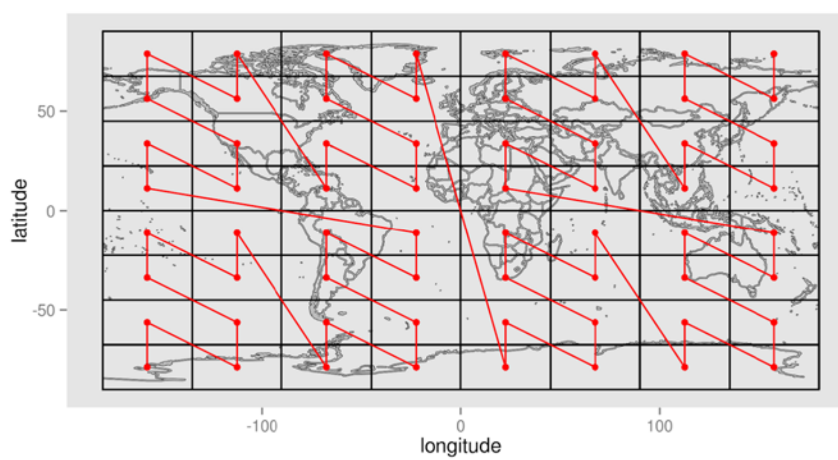
GeoMesa – Spatiotemporal Store

K-V encoding tricks can be arbitrarily complex

KEY							**VALUE**
ROW			**COLUMN**		**TIMESTAMP**	**VIZ**	
			COLUMN FAMILY	**COLUMN QUALIFIER**			
Epoch Week *2 bytes*	**Z3(x,y,t)** *8 bytes*	Unique ID (such as UUID)	"F"	-	-	Security tags	Byte-encoded **SimpleFeature**

Hierarchical Storage

Hash Functions



+ Secondary or Full Indexes on Values

Why use K-V stores?

- Scale out – large commodity clusters
- Transparent partitioning, layout, performance
  - Friendly to developers & applications that want control
- Some have "special features"
  - entry-level access control
- Simple solution, *if* all your application requires is to <u>read & write entries by key</u>

Why use K-V stores?

- Scale out – large commodity clusters
- Transparent partitioning, layout, performance
  - Friendly to developers & applications that want control
- Some have "special features"
  - entry-level access control
- Simple solution, *if* all your application requires is to <u>read & write entries by key</u>

What if you want to do more?

Computing beyond read-writes

Computing @application client can have drawbacks…

Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

DATA TYPE	KEY	VALUE
Session	User/Session ID	Session Data (pages visited, shopping cart, …)

> **Fast read-writes perfect for low-latency web server**

> **What if the website managers want to run analytics?**

Simple filters, aggs

Requires re-shuffle

Requires iteration

ML, matrix, graph alg.

Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

DATA TYPE	KEY	VALUE
Session	User/Session ID	Session Data (pages visited, shopping cart, …)

> **Fast read-writes perfect for low-latency web server**

> **What if the website managers want to run analytics?**

Simple filters, aggs

Requires re-shuffle

Requires iteration

ML, matrix, graph alg.

Query: find average session length.

✓ Supported by most key-value databases

Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

DATA TYPE	KEY	VALUE
Session	User/Session ID	Session Data (pages visited, shopping cart, …)

> **Fast read-writes perfect for low-latency web server**

> **What if the website managers want to run analytics?**

Simple filters, aggs

Requires re-shuffle

Requires iteration

ML, matrix, graph alg.

Query: find average session length.

Query: for each page, find average time spent viewing it.

Unsupported by most, unless you build an index

Need an external system

Computing on Key-Value DB

Example from Riak Use Case: Browser Sessions (e.g. Wikipedia)

DATA TYPE	KEY	VALUE
Session	User/Session ID	Session Data (pages visited, shopping cart, …)

> **Fast read-writes perfect for low-latency web server**

> **What if the website managers want to run analytics?**

Simple filters, aggs

Requires re-shuffle

Requires iteration

ML, matrix, graph alg.

Query: find average session length.

Query: for each page, find average time spent viewing it.

Query: build a histogram of users browsing certain pages, grouped by session age, over various time periods.

Query: group users into clusters based on pages browsed. Recommend users to browse popular pages browsed in their cluster.

Computing on Key-Value DB

State of the Art

Simple filters, aggs	Supported inside K-V DBs, to various extents
Requires re-shuffle	
Requires iteration	
ML, matrix, graph alg.	or: Pull data into an external serial (application client) or distributed (MapReduce, Spark, Presto, …) system

Computing on Key-Value DB

This Talk

Simple
filters, aggs

Requires
re-shuffle

Requires
iteration

ML, matrix,
graph alg.

New support
inside K-V DBs

GRAPHULO

RACO

or: Pull data into an external serial (application client) or distributed (MapReduce, Spark, Presto, …) system

> **How to implement analytics in K-V DBs?**
> **When is it good to do so?**
> – **Consider: # of re-shuffles, access path, selectivity, …**

Computing on Key-Value DB

Benefits of Server-Side Computation

1. **Data Locality**
   - Save communication
2. **Reuse infrastructure**
   - One less system to adopt and maintain
3. **Database features for free**
   - Indexed access
   - Distributed execution

Graphulo: Linear Algebra for Apache Accumulo

or, conceptually, any
Google BigTable-based K-V DB

UNIVERSITY *of* WASHINGTON

GraphBLAS: Working Spec for LA

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

for *Sparse* Matrices

Matrix as K-Vs

Key					Value
Row	Column			Timestamp	
	Family	Qualifier	Visibility		

> Adjacency list view of Matrix

> (Row, Column Qualifier, Value)
= $(v_1, v_2, \text{weight})$
[Transpose: $(v_2, v_1, \text{weight})$]

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

Example K-Vs

```
row :colq ->val

1 :1 []   -> 141
1 :10 []  -> 12
1 :101 [] ->  9
1 :105 [] ->  3
1 :11 []  ->  9
1 :110 [] ->  3
10 :1 []  -> 18
10 :109 [] -> 2
15 :1 []  -> 18
15 :109 [] -> 2
```

Other schemas supported:

– **Edge List / Incidence Matrix**
– **Single Table with transpose & degrees**

Matrix as K-Vs

Key					Value
Row	Column			Timestamp	
	Family	Qualifier	Visibility		

- Adjacency list view of Matrix
- (Row, Column Qualifier, Value)
  = (v_1, v_2, weight)
  [Transpose: (v_2, v_1, weight)]

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

Other schemas supported:
- Edge List / Incidence Matrix
- Single Table with transpose & degrees

Partition by Row

Tablet 1

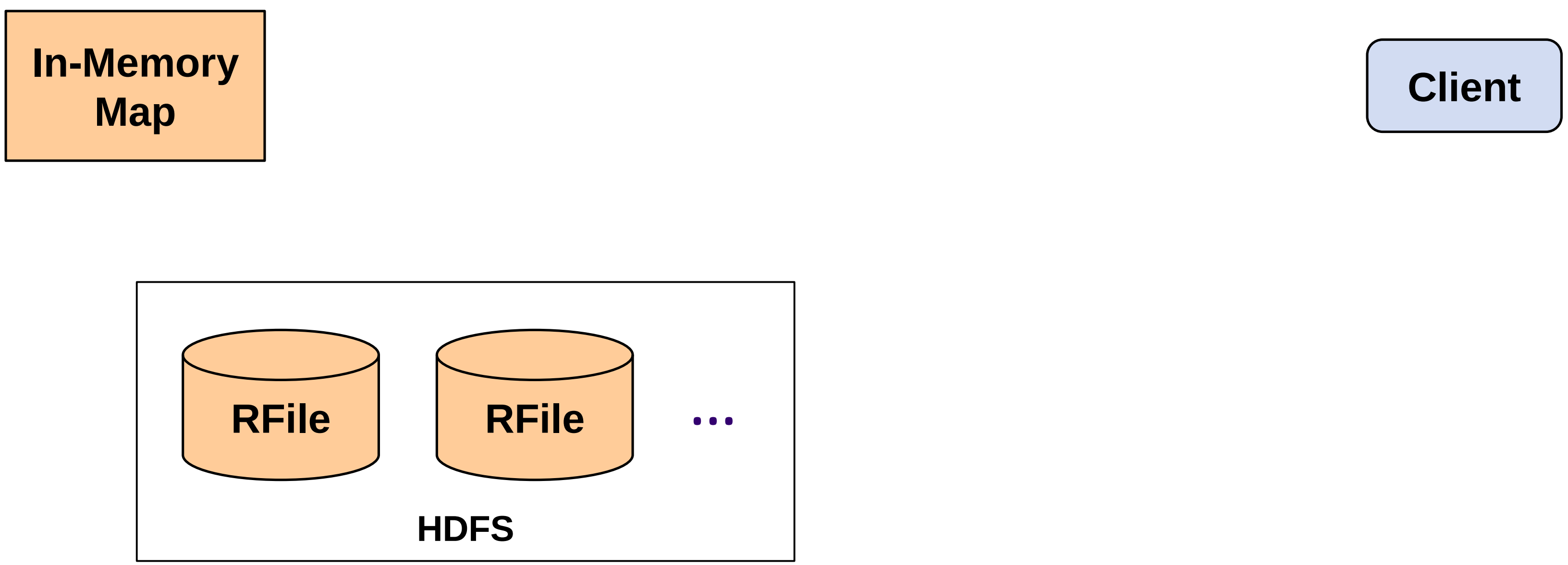
Tablet 2

Example K-Vs

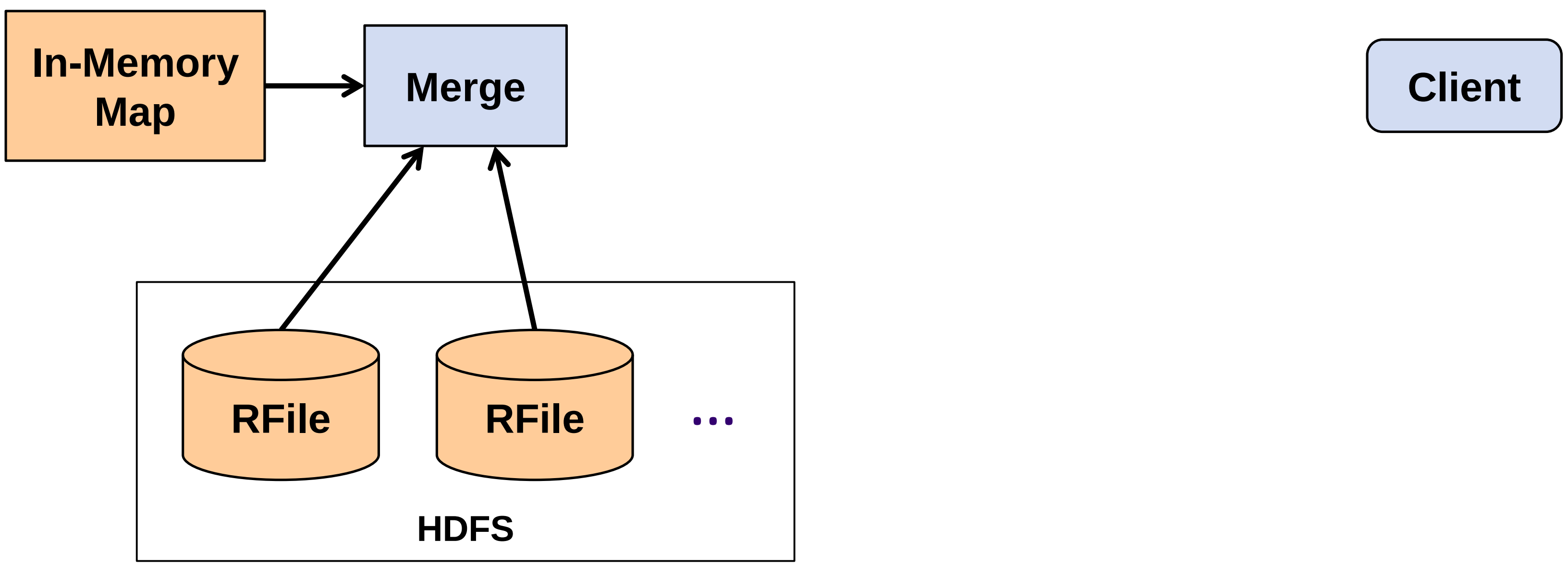
```
row :colq ->val

1 :1 []   -> 141
1 :10 []  -> 12
1 :101 [] ->  9
1 :105 [] ->  3
1 :11 []  ->  9
1 :110 [] ->  3
10 :1 []  -> 18
10 :109 [] -> 2
15 :1 []  -> 18
15 :109 [] -> 2
```

Accumulo Scan Iterator Pipeline

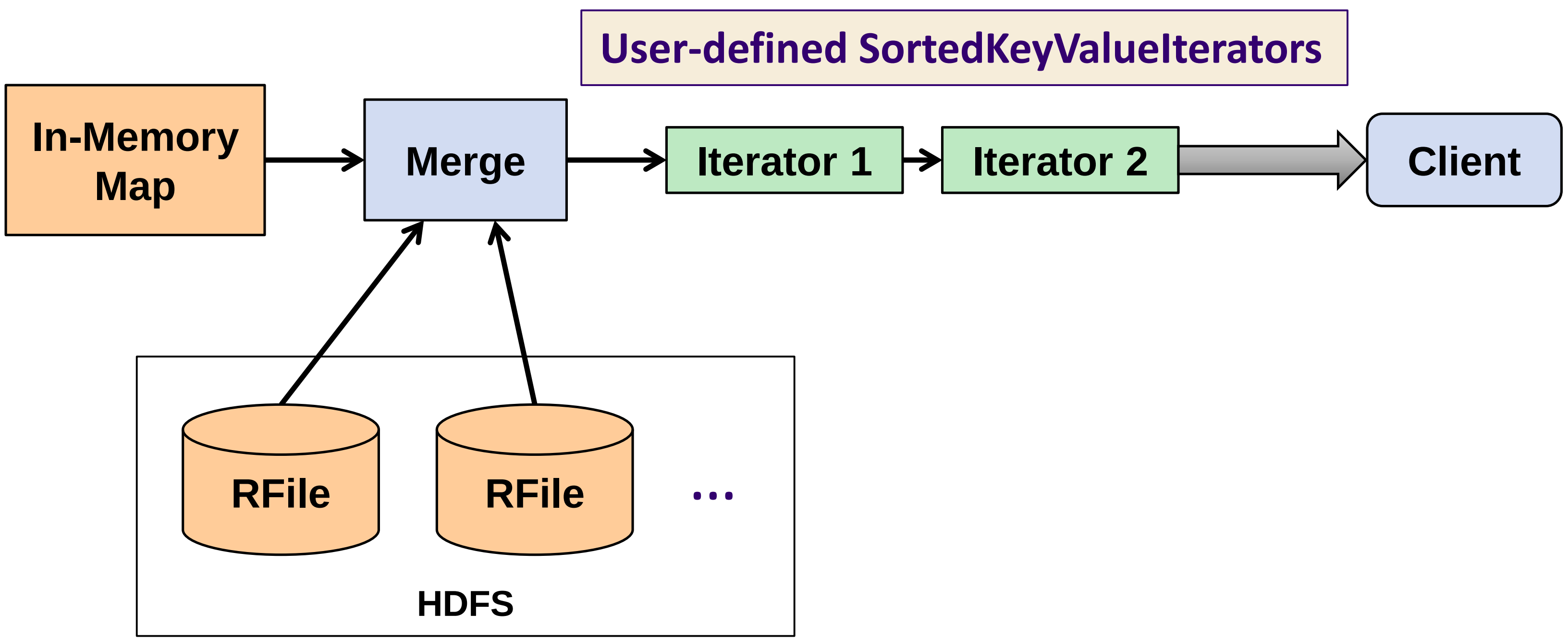
Goal: Understand Accumulo's support for in-database computation in order to re-purpose it for analytics



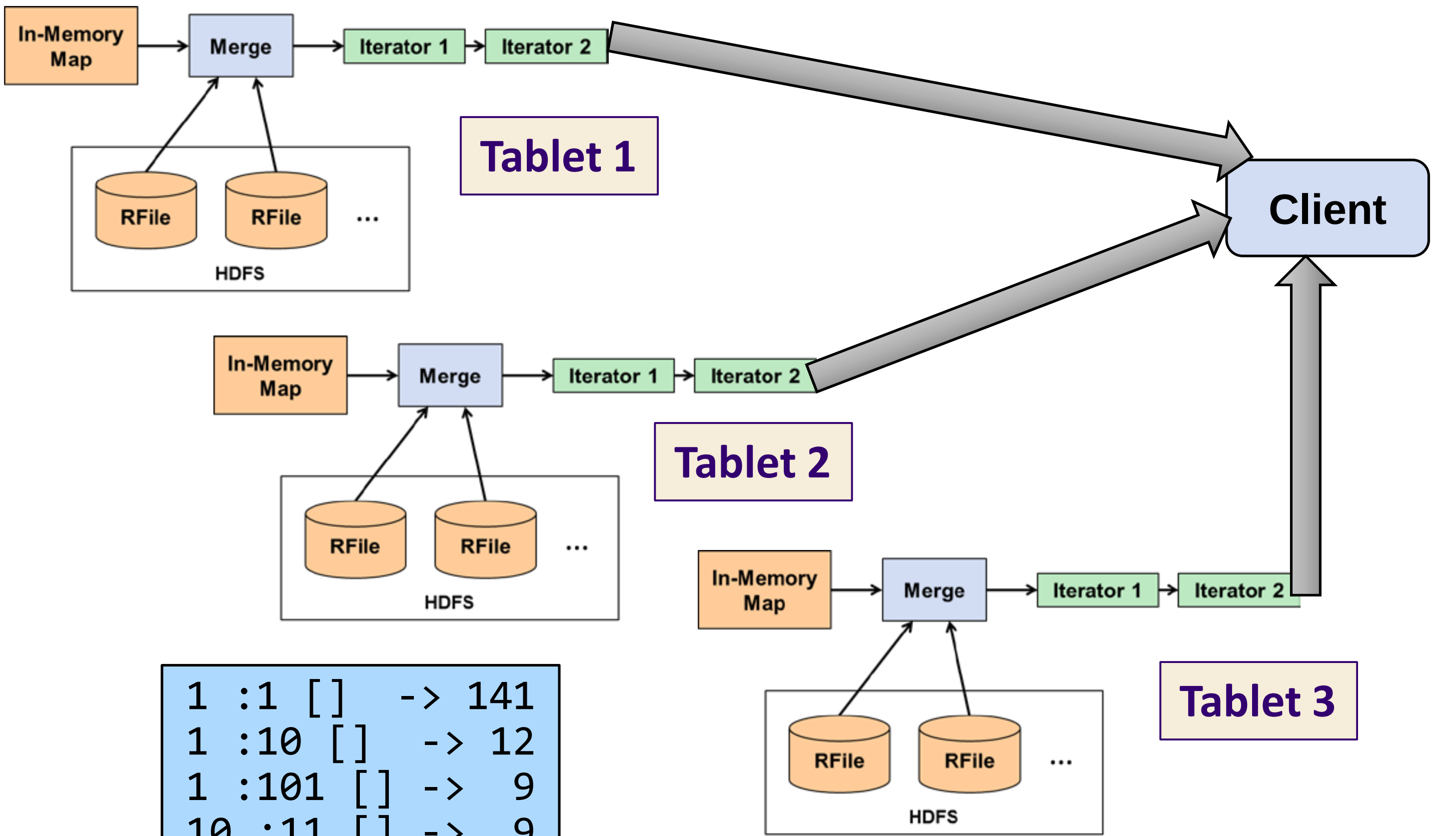
Accumulo Scan Iterator Pipeline



Accumulo Scan Iterator Pipeline

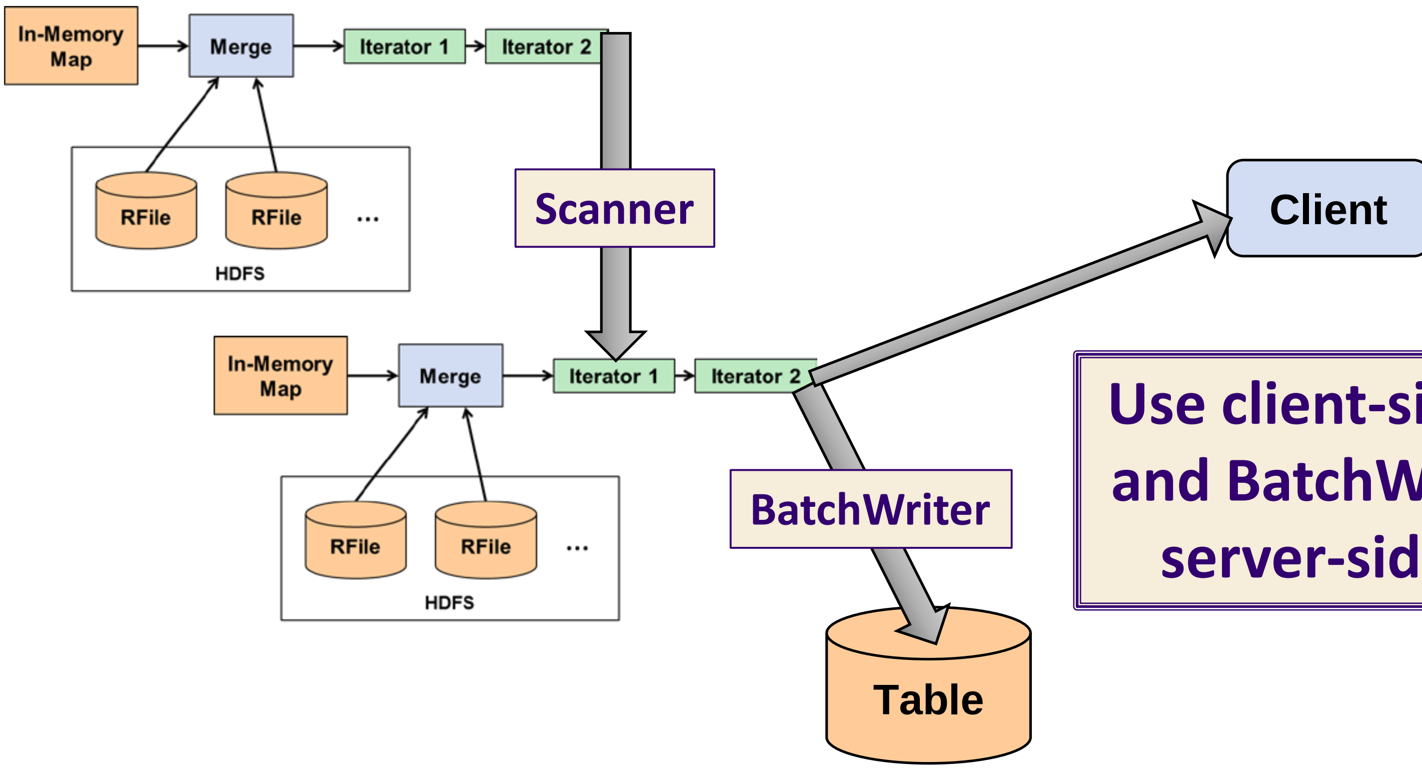


Accumulo BatchScan Iterator Pipeline



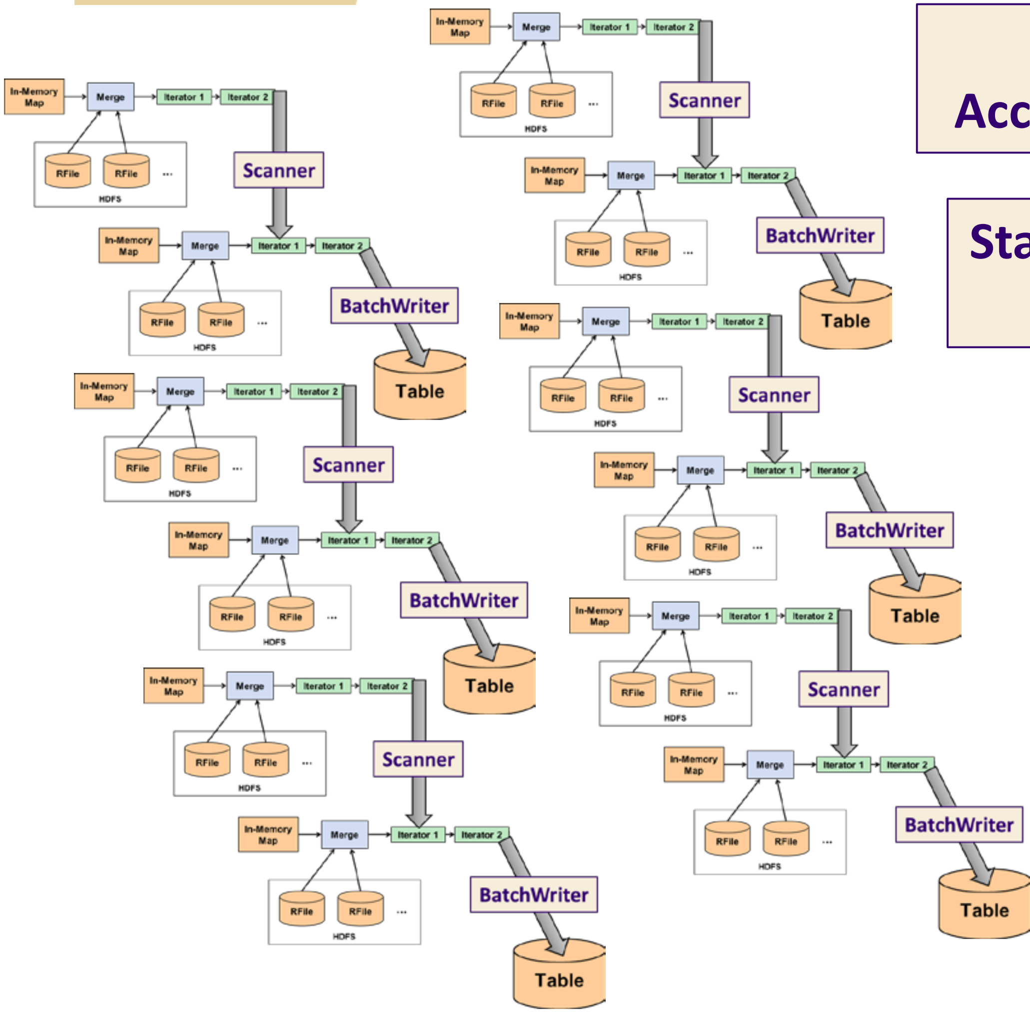

```
1  :1  []   -> 141
1  :10 []   -> 12
1  :101 [] ->  9
10  :11 [] ->  9
---------------
15  :1  []   -> 18
15  :109 [] -> 2
```

Graphulo addition to Iterator Pipeline



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

Graphulo addition to Iterator Pipeline

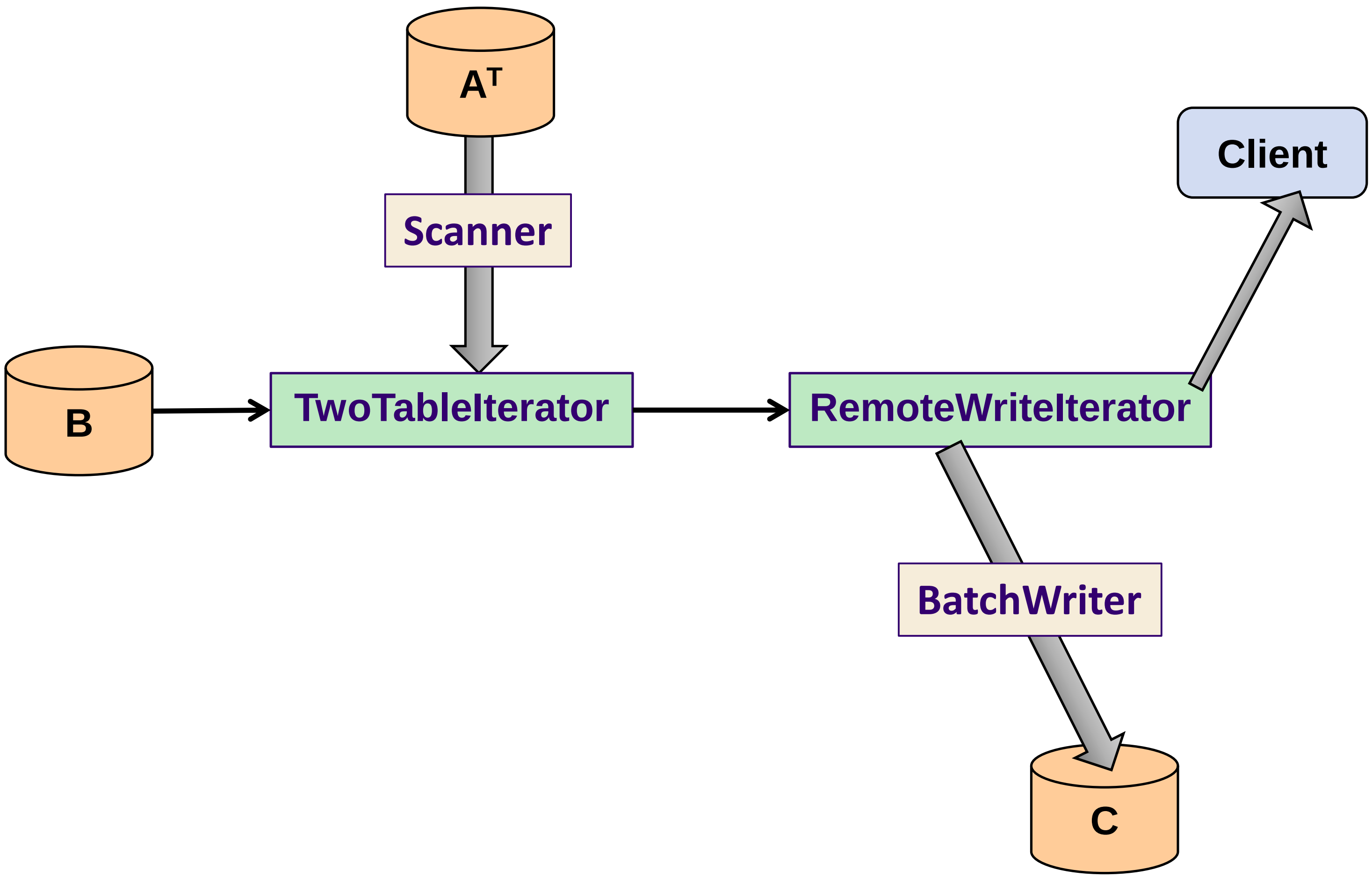


Distributes with Accumulo's Tablet Servers

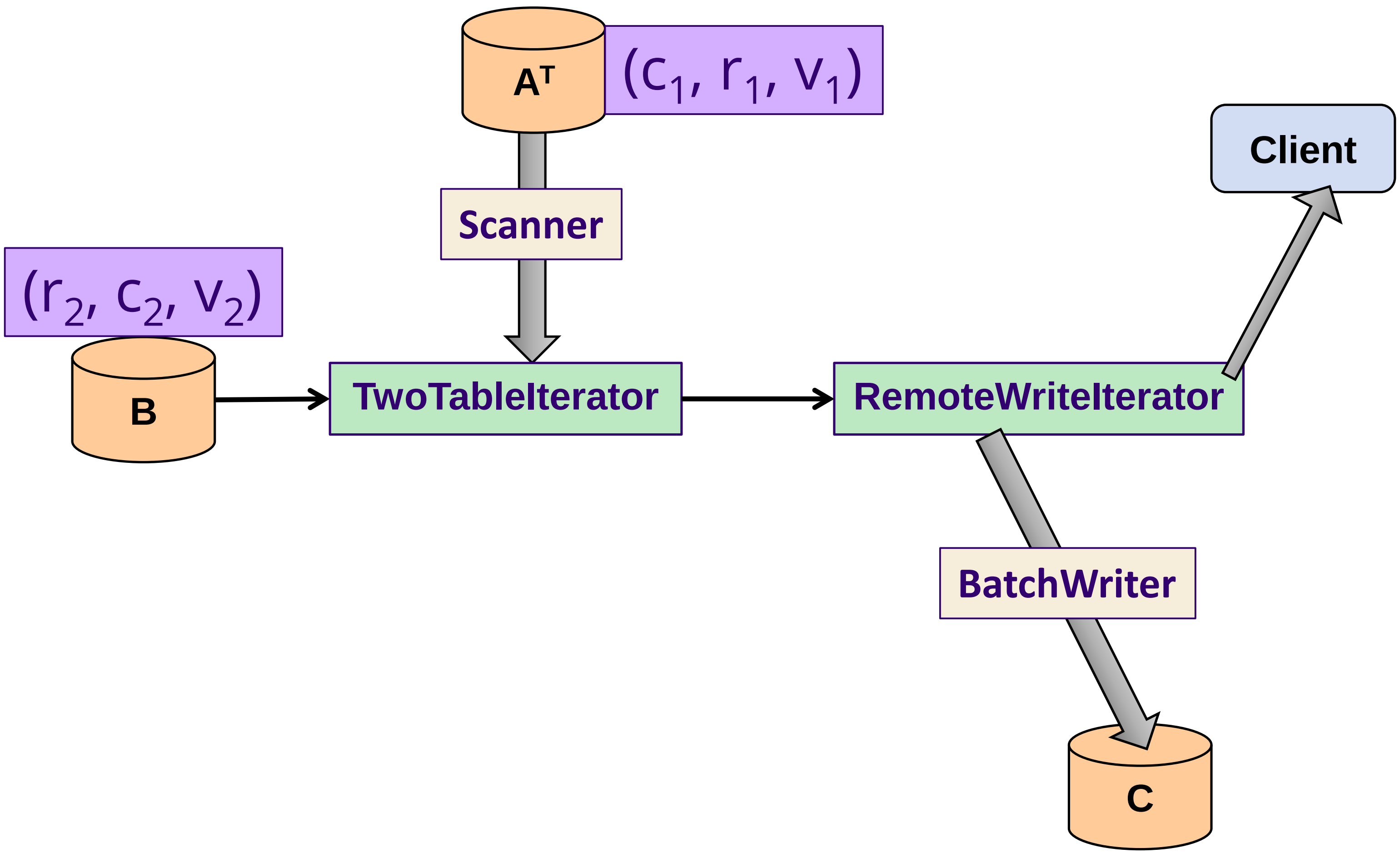
Standing tablet server thread pools service queries interactively

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

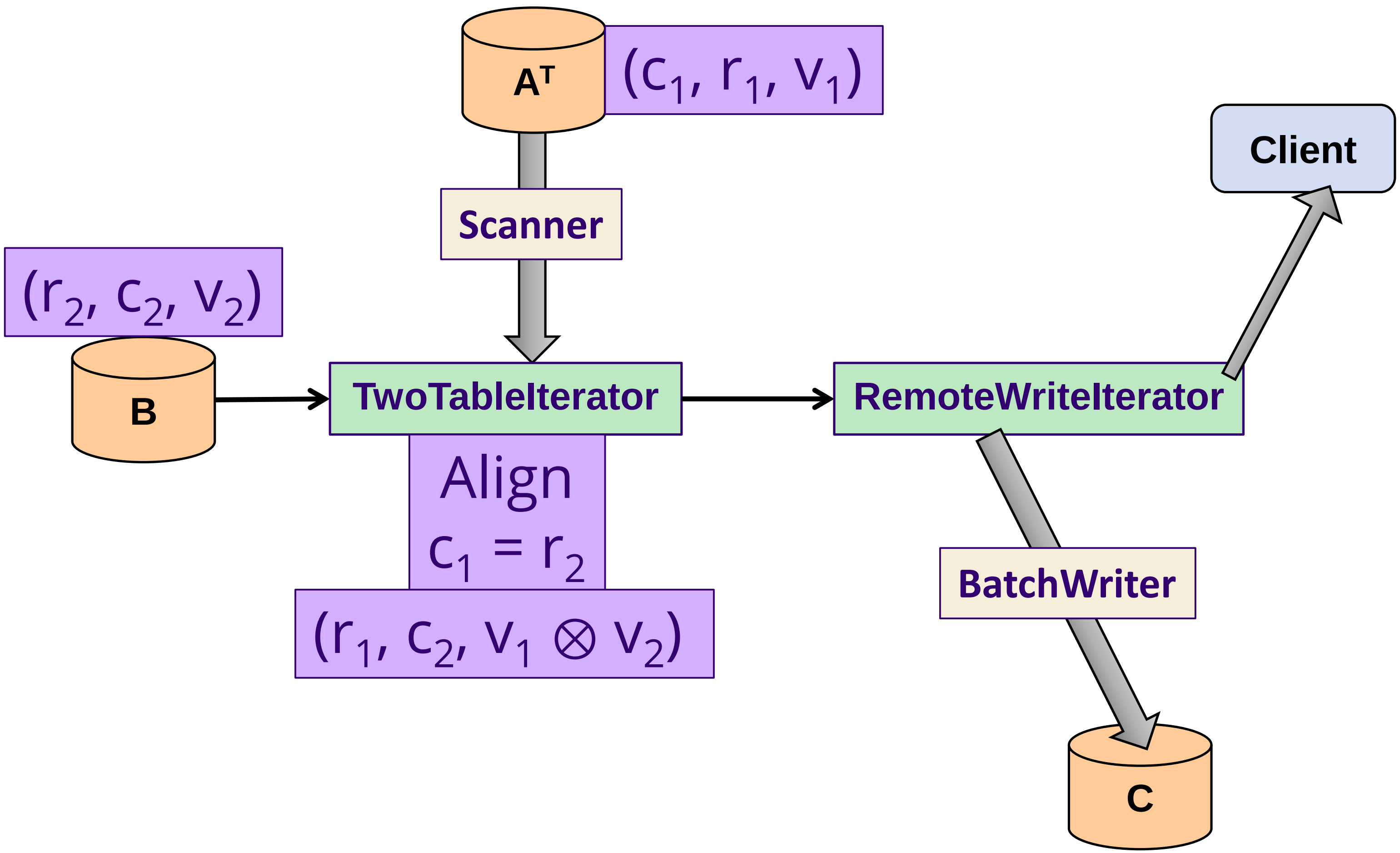
Graphulo MxM: $A^T \,(\oplus . \otimes)\, B$



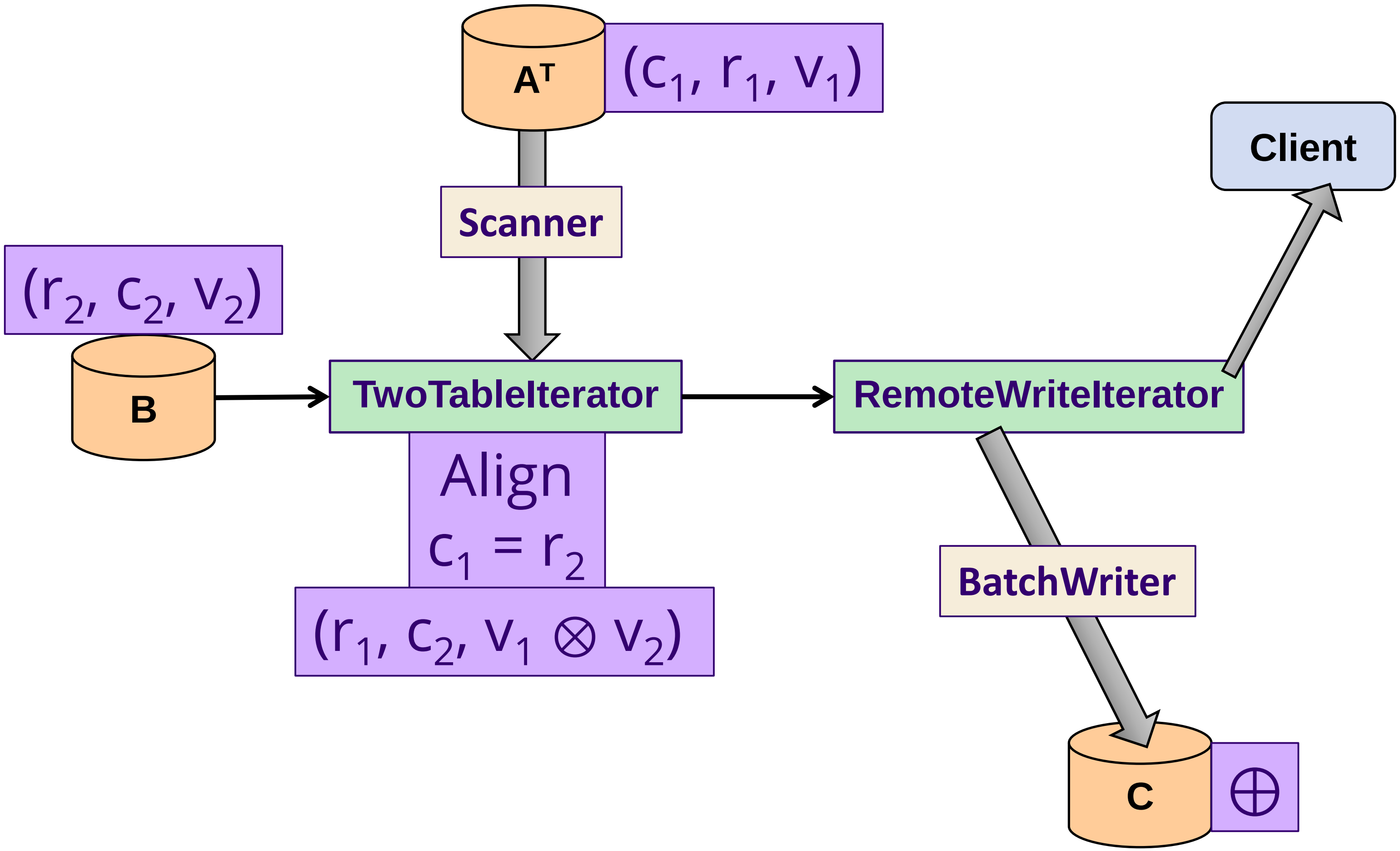
Graphulo MxM: $A^T (\oplus . \otimes) B$



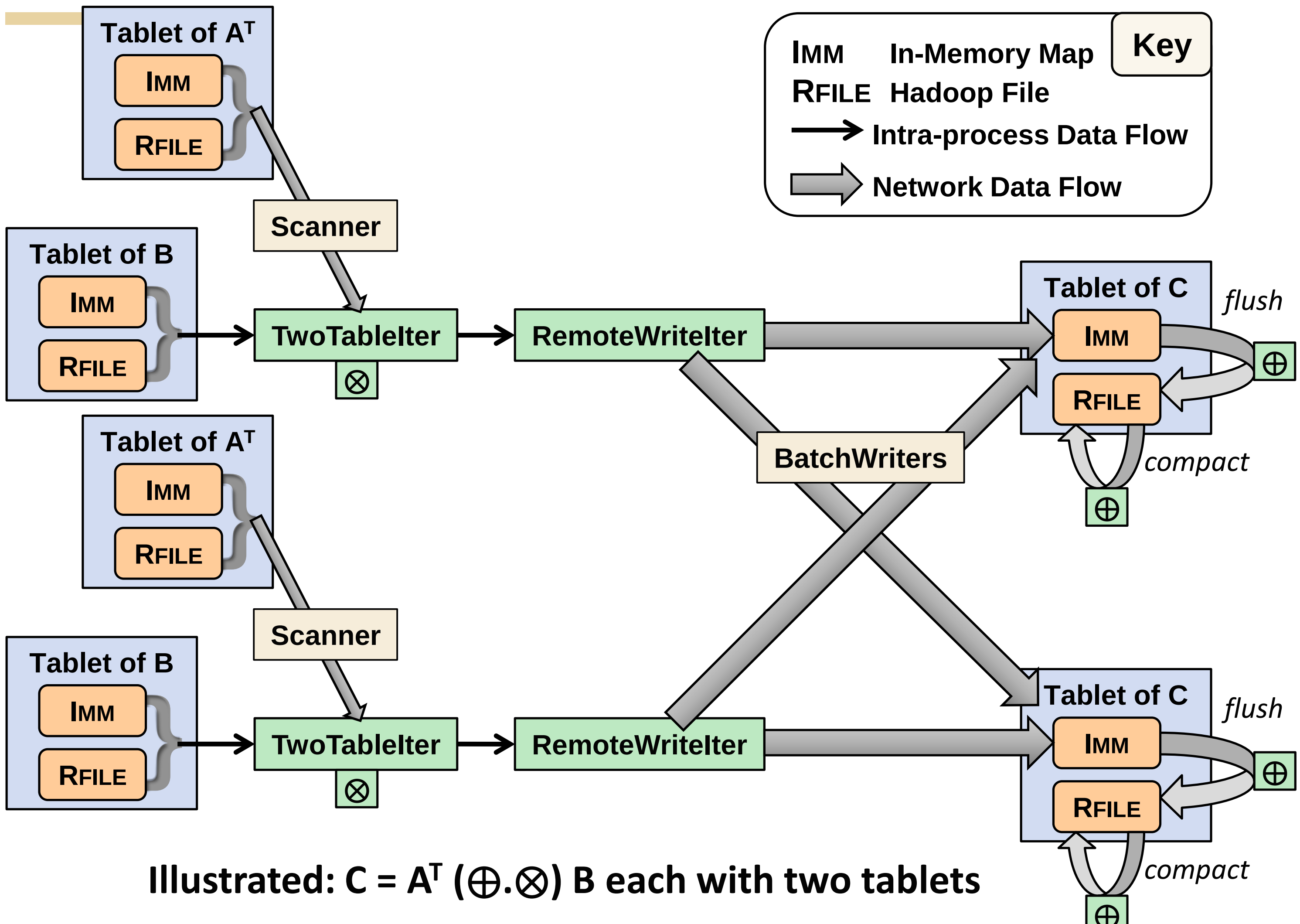
Graphulo MxM: $A^T (\oplus . \otimes) B$



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



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



Illustrated: C = A^T ($\oplus.\otimes$) B each with two tablets

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)
…
```

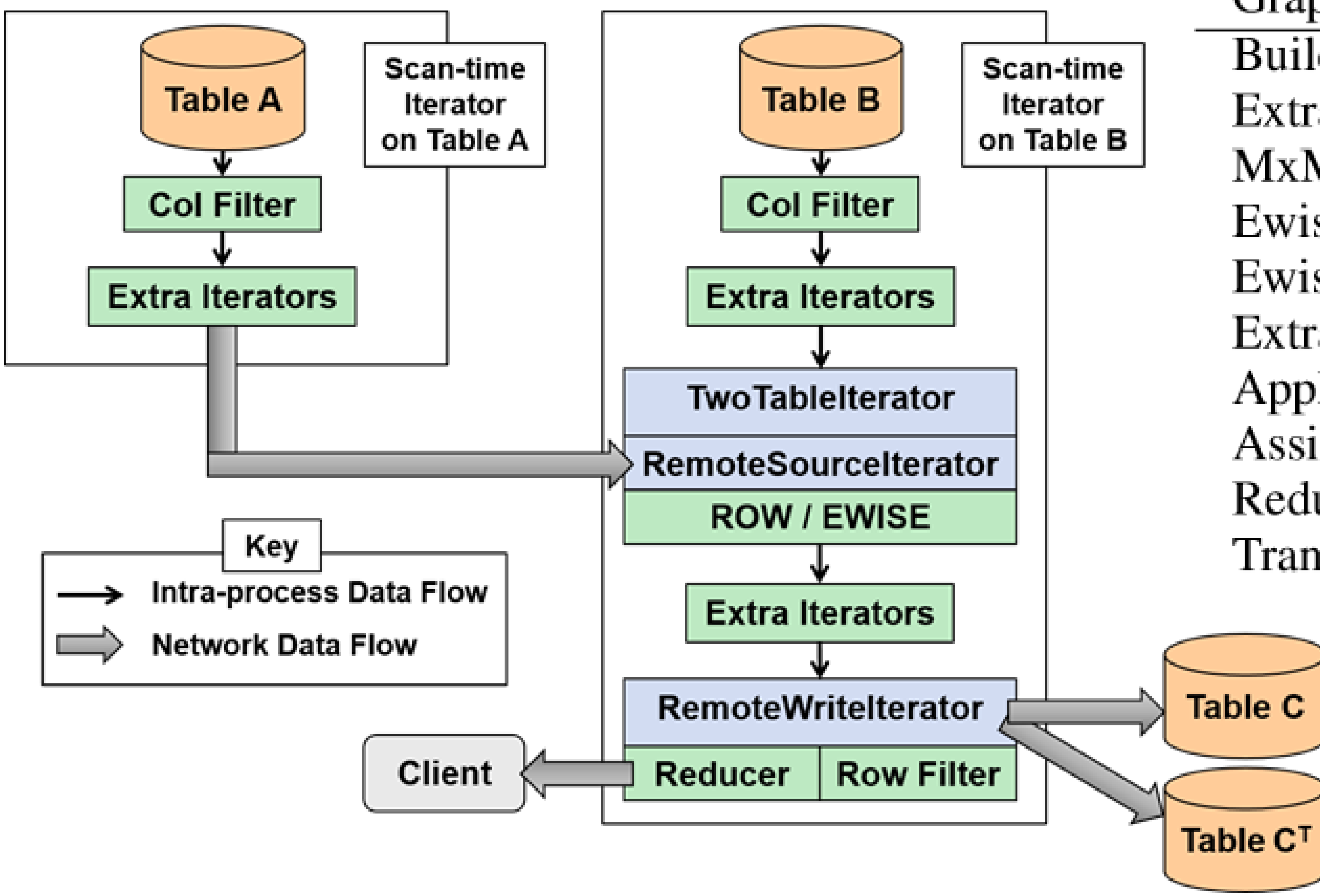
Simple API abstracts the iterator pipeline

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

Evaluating Graphulo

When is it better to use Graphulo vs. an external system?

Past Experiments

Compare Graphulo to

- Single-node in-memory matrix libraries
  - D4M Sparse matrix library (MATLAB)
  - MTJ Dense matrix library (Java)
- Itself
  - Benchmark of Graphulo scalability with cluster size

Graphulo Implementation of Server-Side Sparse Matrix Multiply in the Accumulo Database

2015 HPEC

Dylan Hutchison,†§* Jeremy Kepner,†‡◇* Vijay Gadepally,†‡* Adam Fuchs+

†MIT Lincoln Laboratory, §University of Washington,
‡MIT Computer Science & AI Laboratory, ◇MIT Mathematics Department, +Sqrrl, Inc.

Graphulo faster than single-node in-memory external systems on I/O-bound, single-pass computation

The GraphBLAS standard provides a compact and efficient basis for a wide range of graph applications through a small number

tiply (SpGEMM), the core kernel at the heart of GraphBLAS.

- Task: Matrix Multiply (MxM)
- Graphulo vs. D4M, Single-node
- Result: Graphulo universally faster
- Why?
  - I/O costs dominate (locality is good)
  - MxM can be formulated as a single-pass algorithm (use outer product)

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

2016 HPEC

Dylan Hutchison[†] Jeremy Kepner[‡§◇] Vijay Gadepally[‡§] Bill Howe[†]

[†]University of Washington [‡]MIT Lincoln Laboratory
[§]MIT Computer Science & AI Laboratory [◇]MIT Mathematics Department

Best Student Paper

Results hold for more complex graph algorithms

- Tasks: Find Jaccard coefficients (vertex similarity), k-Truss subgraph (community detection)
- Graphulo vs. D4M vs. MTJ
- Result: Graphulo universally faster at Jaccard; D4M/MTJ universally faster at k-Truss
- Why?
  - Jaccard can be expressed as a fused MxM
  - k-Truss iterations require one pass each
  - Accumulo writes intermediary tables to disk

UNIVERSITY *of* WASHINGTON

Benchmarking the Graphulo Processing Framework

2016 HPEC

Timothy Weale[†], Vijay Gadepally[‡§*], Dylan Hutchison[°] and Jeremy Kepner[‡§+]
[†]Department of Defense, [‡]MIT Lincoln Laboratory, [§]MIT Computer Science & AI Laboratory
[+]MIT Mathematics Department, [°]University of Washington

***Abstract*—Graph algorithms have wide applicablity to a variety of domains and are often used on massive datasets. Recent standardization efforts such as the GraphBLAS specify a set of key computational kernels that hardware and software developers can adhere to. Graphulo is a processing framework that enables GraphBLAS kernels in the Apache Accumulo databas Graphu multipli present the results of scaling the Graphulo engine to larger problems and scalablity when a greater number of resources is used. Specifically, we present two experiments that demonstrate Graphulo scaling performance is linear with the number of available res processing ra large graphs experiment u a large grap These bench who wish to**

is provided in [12]. It is natural that a processing framework designed for databases makes use of the GraphBLAS kernels.

Graphulo [13] is a specialized graph processing framework built to work with Apache Accumulo and to conform to the GraphBLAS standard. Apache Accumulo is a NoSQL database], [15]. ulo has settings.

As shown in Figure 1, Accumulo's data model has a key nn qualifier, ulo provides tex), column number of isibility and paper.

Graphulo scales with Accumulo as cluster size increases

- **Tasks: MxM, Subgraph Extraction**
- **Graphulo Scalability Benchmark**
- **Results: Linear weak & strong scaling on MxM. Subgraph extraction scales in interactive range (1-2k edge extraction)**
- **Why?**
  - **Graphulo designed to scale with Accumulo**

Past experiments compare Graphulo to itself & to in-memory single-node external systems.

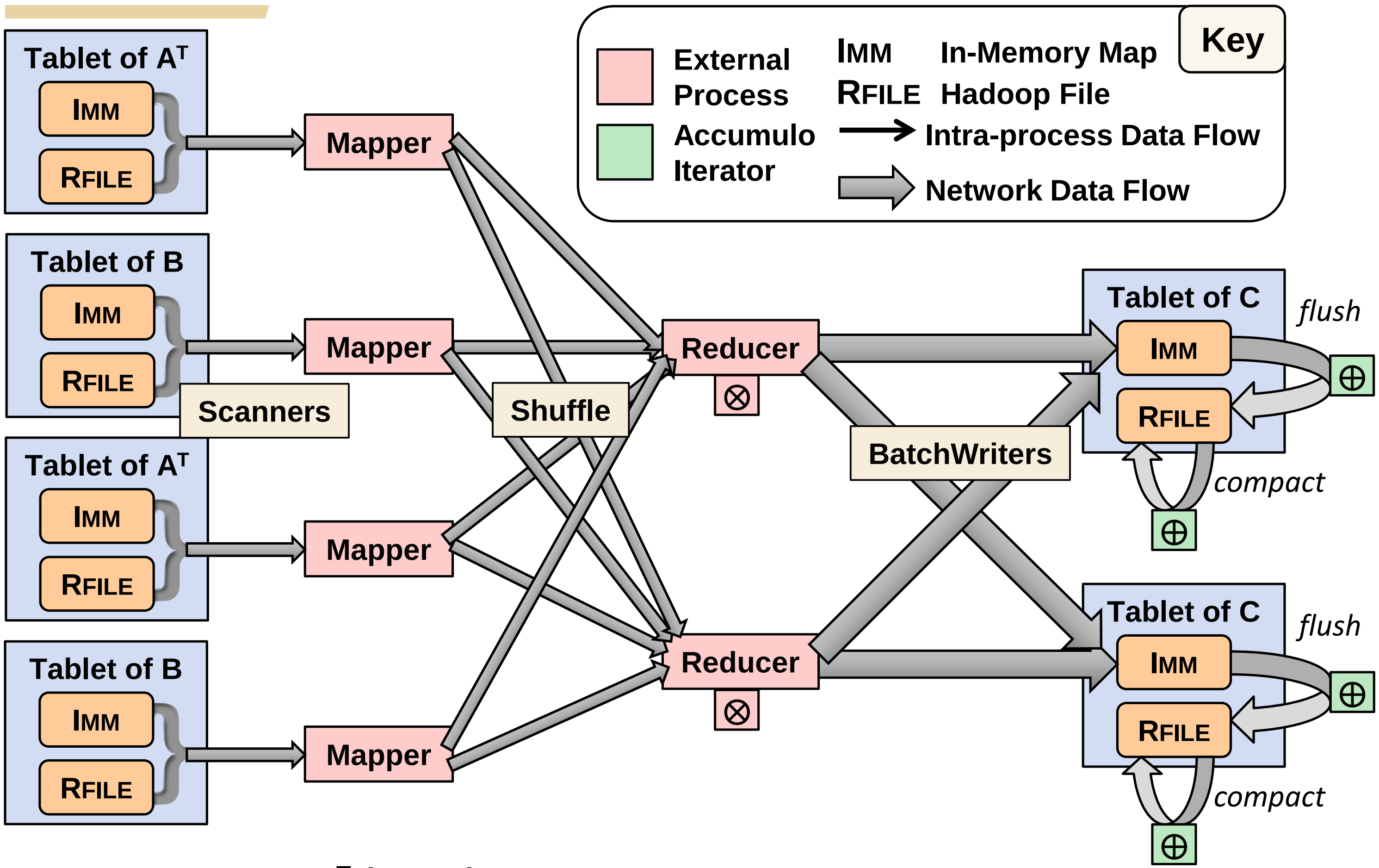
Where's the experiment comparing Graphulo to *distributed* external systems?

(answer: this talk)

MxM: Graphulo vs. MapReduce

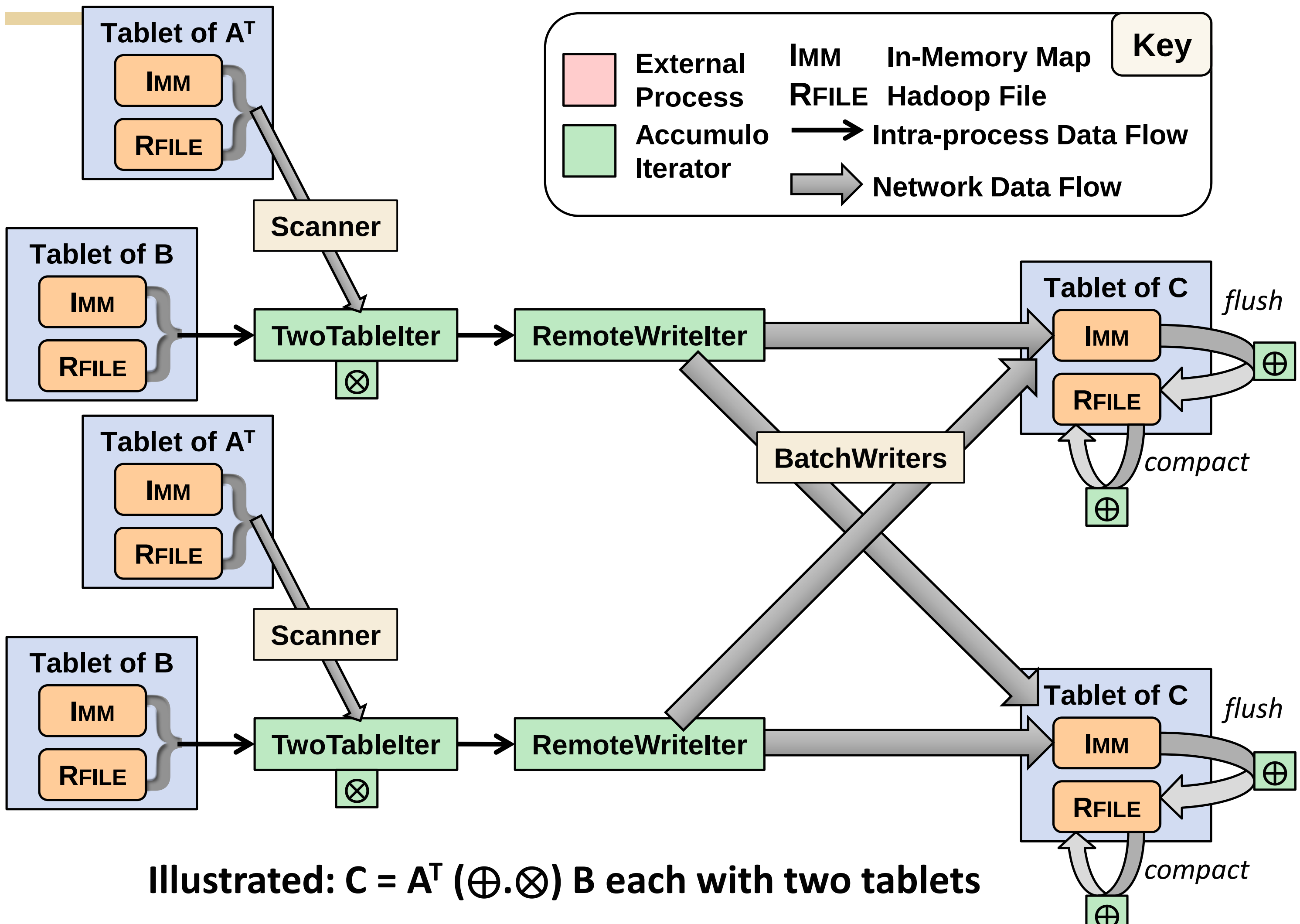
- MapReduce well supported by Accumulo
  - "Use Accumulo for low-latency queries on subgraphs"
  - "Use MapReduce for high-throughput analytics"
  - Are these assumptions true?
- MapReduce $C = A^T B$ *(see next slide)*
  - Map phase: BatchScan & Join A^T, B on their common key
  - Reduce phase: Multiply, BatchWrite partial products
    - # of reducers not a major factor
    - Disable speculative execution for correctness
  - Sum partial products via iterator on C
  - Very similar to Graphulo code, except more room to optimize (earlier projection, loop fusion, fewer copies)

MapReduce $C = A^T (\oplus.\otimes) B$



Illustrated: $C = A^T (\oplus.\otimes) B$ each with two tablets, two Reducers

Graphulo $C = A^T (\oplus.\otimes)\ B$ (reminder)



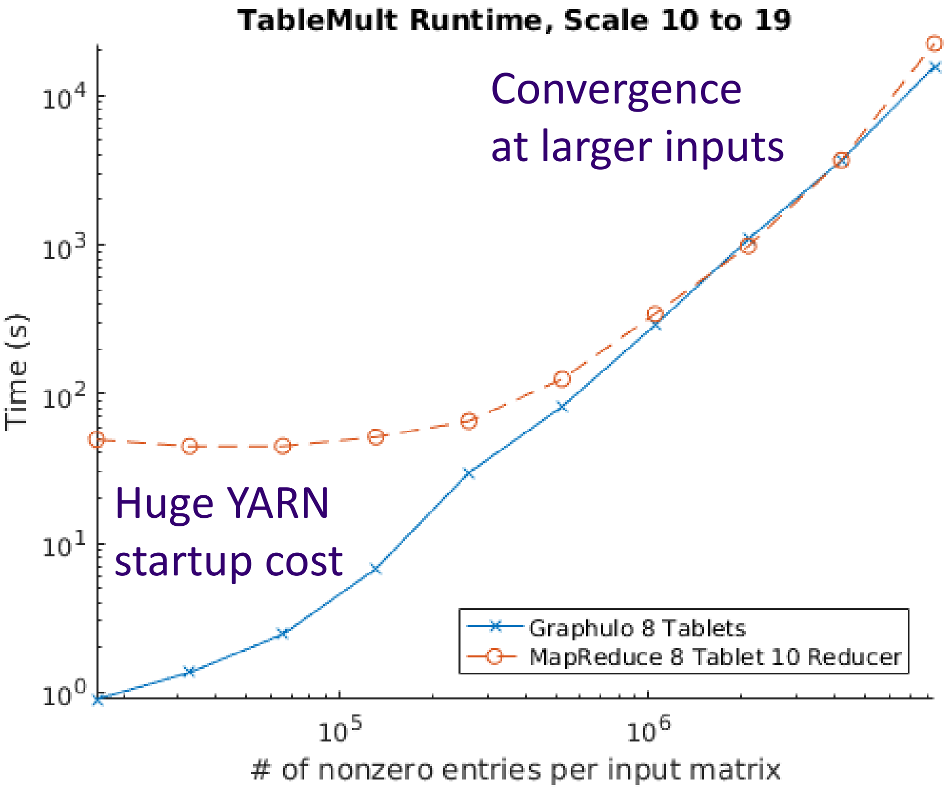
Illustrated: $C = A^T (\oplus.\otimes)\ B$ each with two tablets

MxM: Graphulo vs. MapReduce

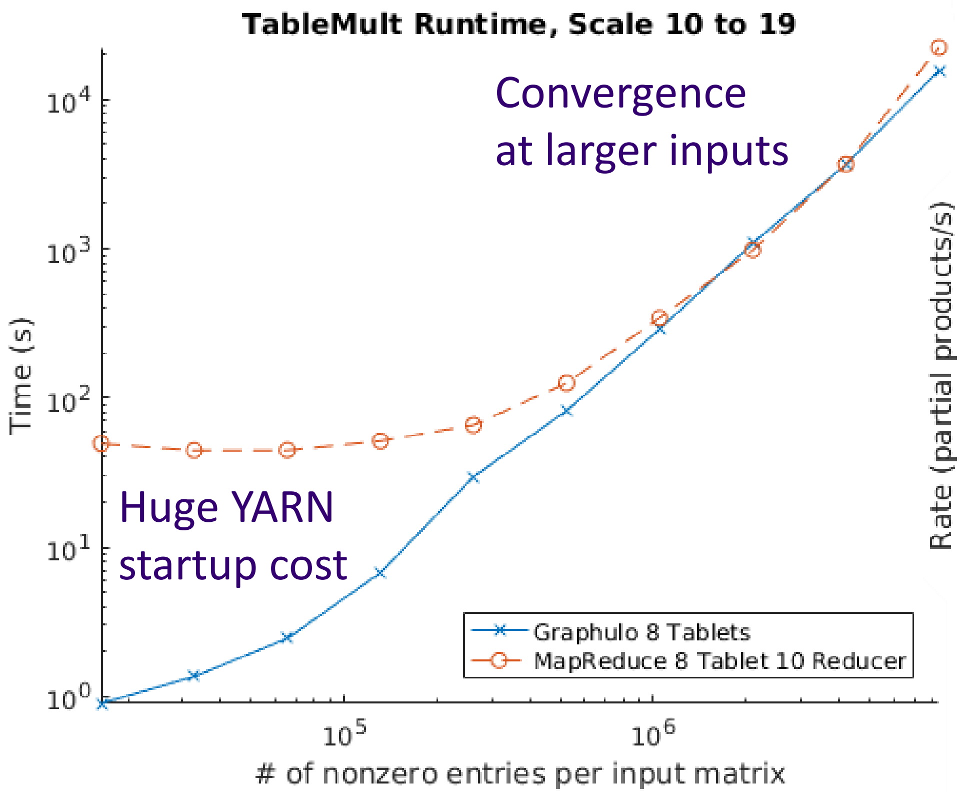
Hardware & Setup

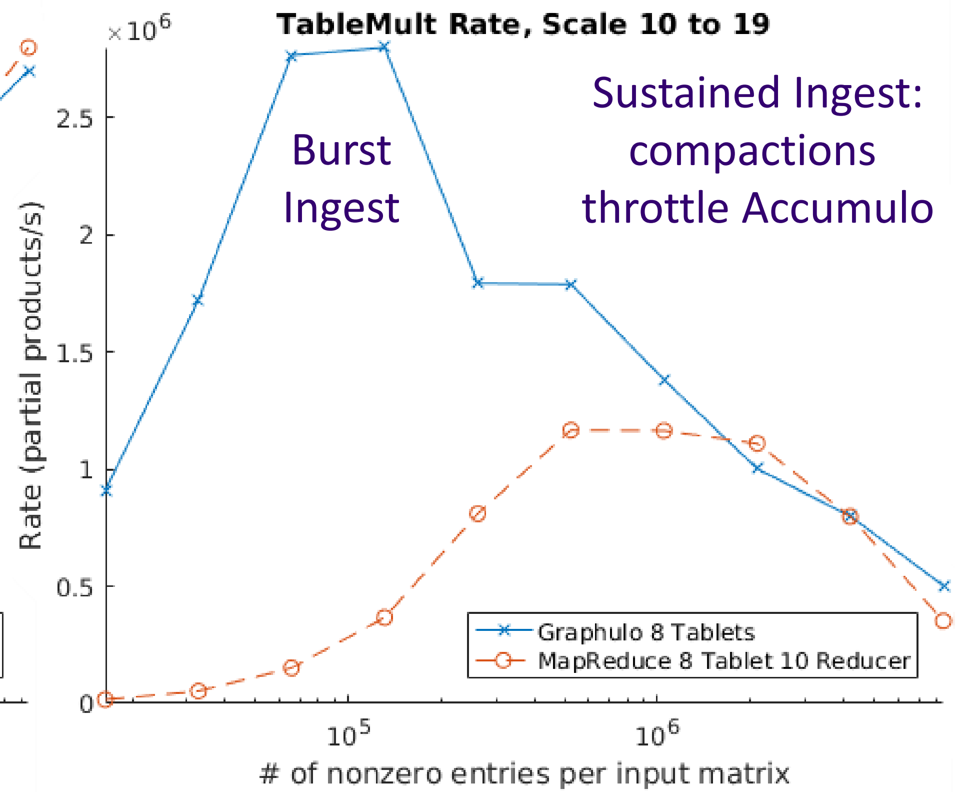
- 12 x m3.large nodes on Amazon EC2, $38/day
  - 7.5 GB memory, 2 vCPUs, 30 GB SSD per node
  - 8 worker tablet servers
    - 3 GB memory for Accumulo, 3 GB for YARN
    - One tablet per node (low intra-node parallelism)
  - 3 coordinator nodes (YARN ResourceManager, Accumulo Master, HDFS NameNode, Zookeeper)
  - 1 metric-monitoring node (Grafana with InfluxDB)
- Graph500 Power law matrix generator
  - From 2^{10} to 2^{19} rows, 16 non-zeros/row, in each input
  - Skew!

MxM: Graphulo vs. MapReduce



MxM: Graphulo vs. MapReduce





UNIVERSITY of WASHINGTON

When to use Graphulo?

- Use Graphulo for I/O-bound single-pass analytics
  - Big speedup for smaller problem sizes
  - Equivalent at larger problem sizes
  - Holds against external in-memory processing (D4M, MTJ matrix libraries) & distributed (MapReduce)
- Use an in-memory system
  for CPU-bound or multi-pass analytics
  - k-Truss task: expressed as a graph algorithm
    that loops over GraphBLAS kernels
  - Horrible to write each intermediary table to disk
    - Yet, even HPC systems write *checkpoints*

RACO: Relational Algebra for Apache Accumulo

$\pi \quad \sigma \quad \bowtie \quad \gamma \quad \cup$

UNIVERSITY of WASHINGTON

How to model a relation in Accumulo?

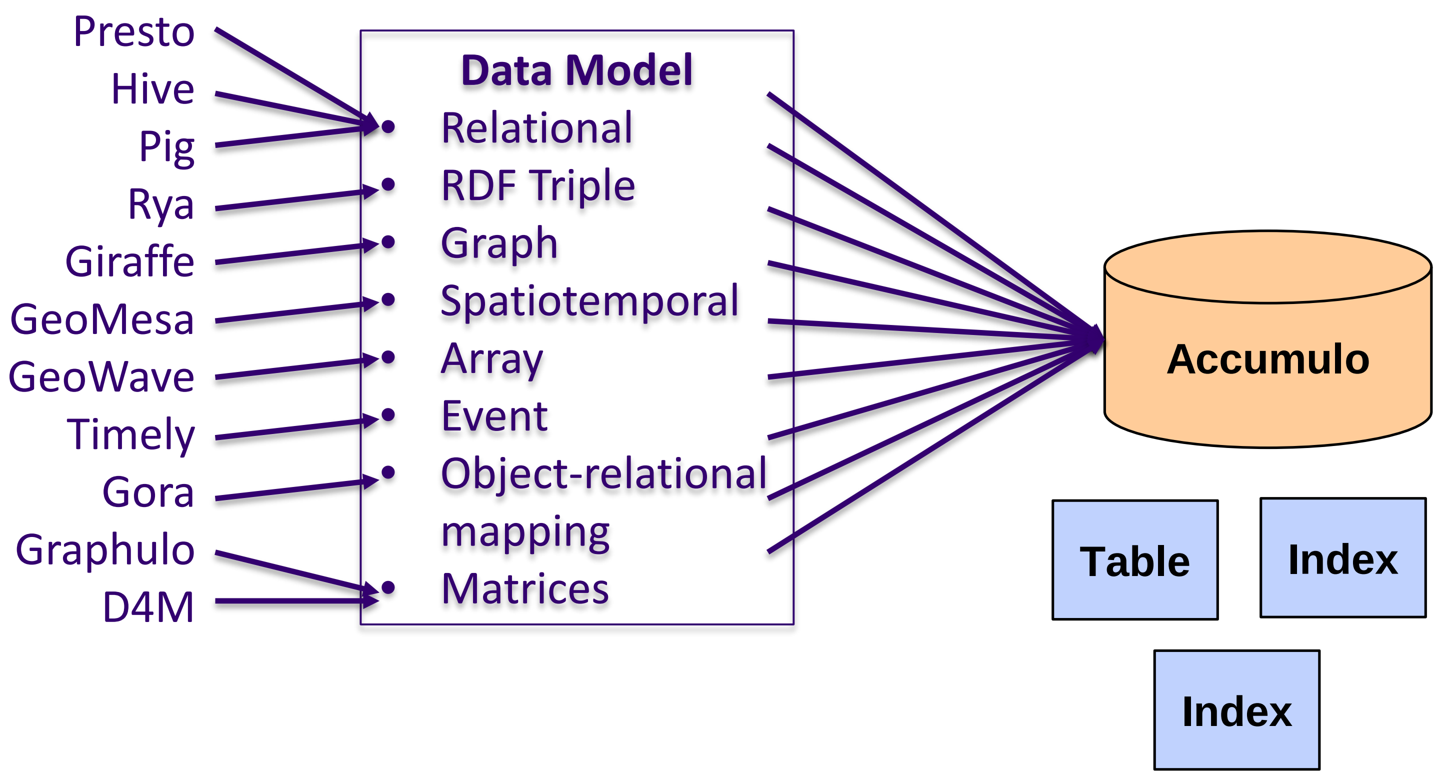
Key					Value
Row	Column			Timestamp	
	Family	Qualifier	Visibility		

More details: public:adhoc:netflow

Name	Type
StartTime	STRING_TYPE
Dur	DOUBLE_TYPE
Proto	STRING_TYPE
SrcAddr	STRING_TYPE
Sport	STRING_TYPE
Dir	STRING_TYPE
DstAddr	STRING_TYPE
Dport	STRING_TYPE
State	STRING_TYPE
sTos	LONG_TYPE
dTos	LONG_TYPE
TotPkts	LONG_TYPE
TotBytes	DOUBLE_TYPE
SrcBytes	LONG_TYPE
Label	STRING_TYPE

> Relations have lots of attributes
> More choices than with matrices
  – Graphulo: adjacency, incidence, single-table schemas
> Choice affects partitioning
> Should work for any Accumulo data

Accumulo Access Paths "in the wild"



Accumulo Access Path "Physical Schema"

For each family, divide attributes into three parts:

DAP | LAP | VAP

DAP: Distributed Access Path

– How data is partitioned across servers; null disallowed

LAP: Local Access Path

– How data is sorted within a server; null disallowed

VAP: Value Access Path

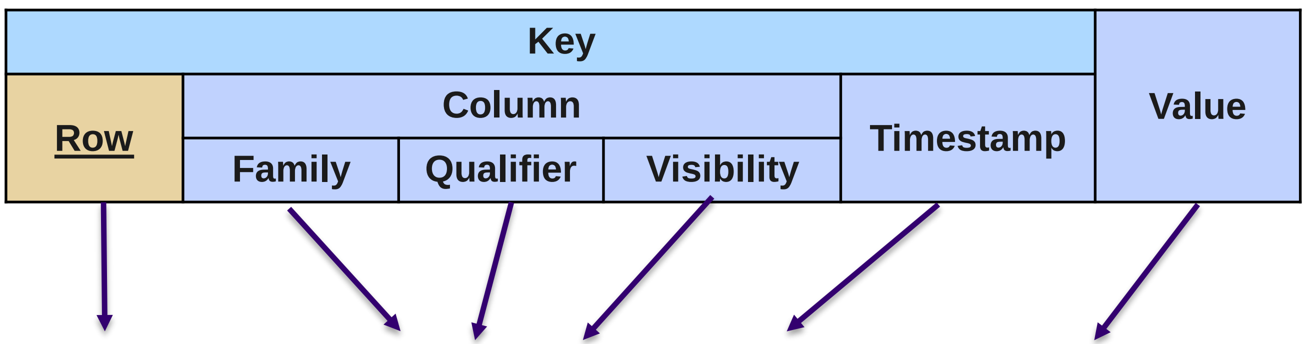
– Other attributes, unsorted; null allowed

<table>
<tr><th colspan="5">Key</th><th rowspan="3">Value</th></tr>
<tr><th rowspan="2"><u>Row</u></th><th colspan="3">Column</th><th rowspan="2">Timestamp</th></tr>
<tr><th>Family</th><th>Qualifier</th><th>Visibility</th></tr>
<tr><td>DAP</td><td>__FAMILY__</td><td>LAP</td><td>VAP__VIS</td><td>VAP__TS</td><td>VAP</td></tr>
</table>

Access Path Netflow Example

DAP | LAP | VAP

TotBytes, StartTime | | SrcAddr, DstAddr, Dur



09:46:59.607825,276	""	""	public	145635001	DstAddr|94.44.127.113
09:46:59.607825,276	""	""	public	145635001	Dur|1.026539
09:46:59.607825,276	""	""	public	145635001	SrcAddr|147.32.84.59

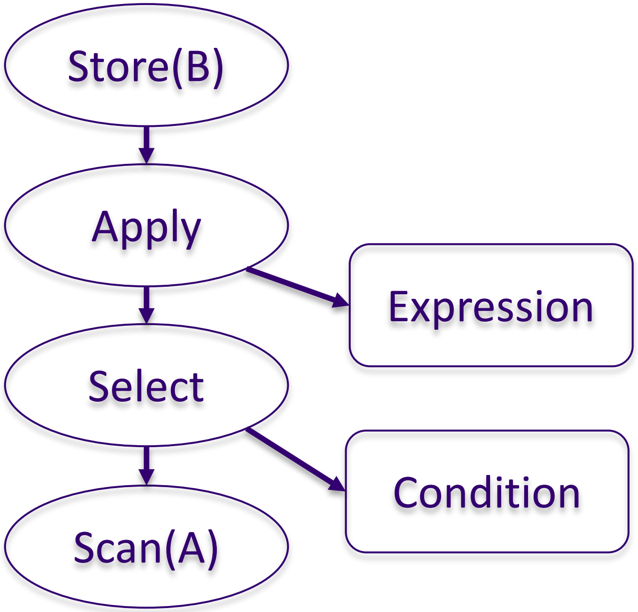
Each tuple is a set of Key-Values

- Facilitates projection, filtering, missing values

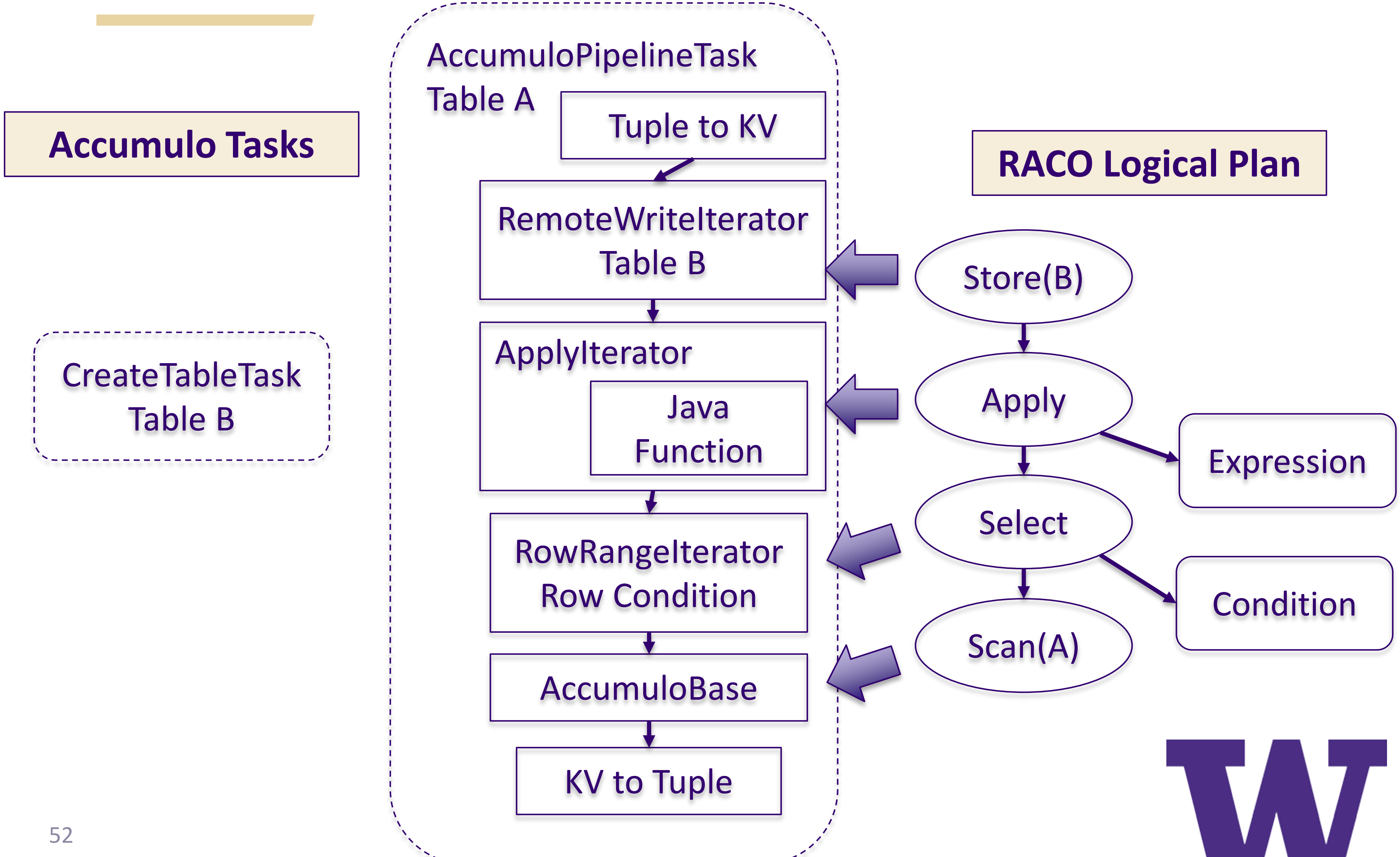
Compiling RA onto Accumulo

Accumulo Tasks

RACO Logical Plan



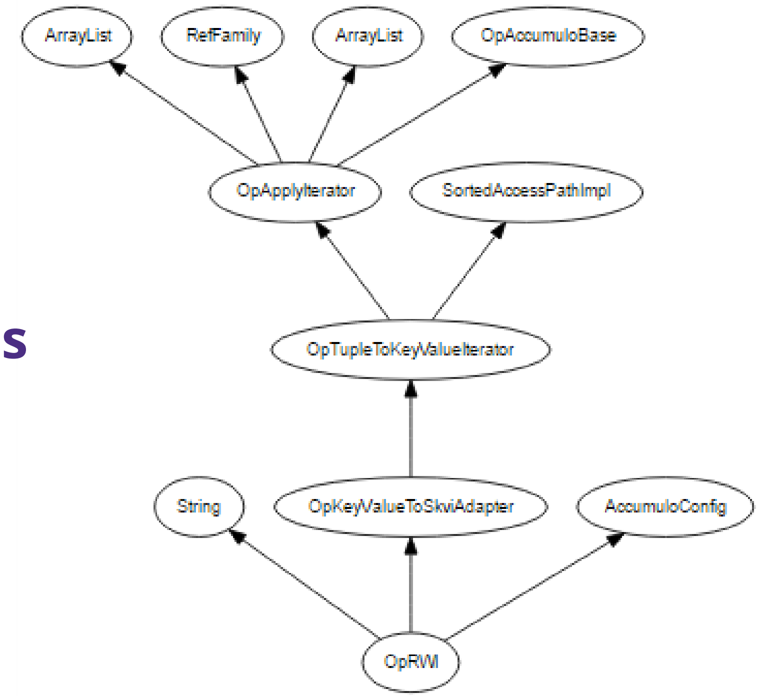
Compiling RA onto Accumulo



Takeaways from RA on Accumulo

Polystore Demo

- RA Compilation used in an October sponsor demo
  - Federated plan executed part Accumulo, part Spark or C
  - Accumulo query stored results in a CSV file for Spark/C
  - "Prototype works!"
    Much more to do…



UNIVERSITY of WASHINGTON

Takeaways from RA on Accumulo

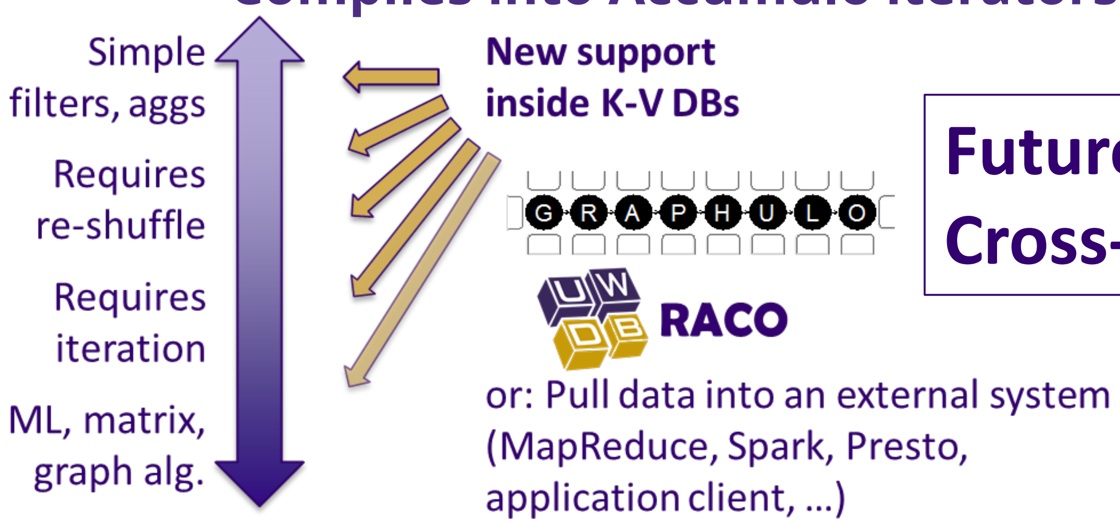
Access Paths hint at a general physical schema

> **Primitives**
  - **Block[n](...): Store contiguous range of n entries together**
  - **Group(...): Each entry stored separately; Block[1]**
  - **Range(...): All entries stored together; Block[N]**

> **Examples**
  - **Accumulo:** Block(row) Group(family) Range(row, family, qualifier, visibility, timestamp) value
  - **Myria hash partitioning:** Group(a_1, ..., a_k) a_1, ..., a_k
  - **TileDB matrix:** Block(j, i) Range(i, j) v
  - **More access paths for indices**
  - **Applications pick both a logical & physical schema**

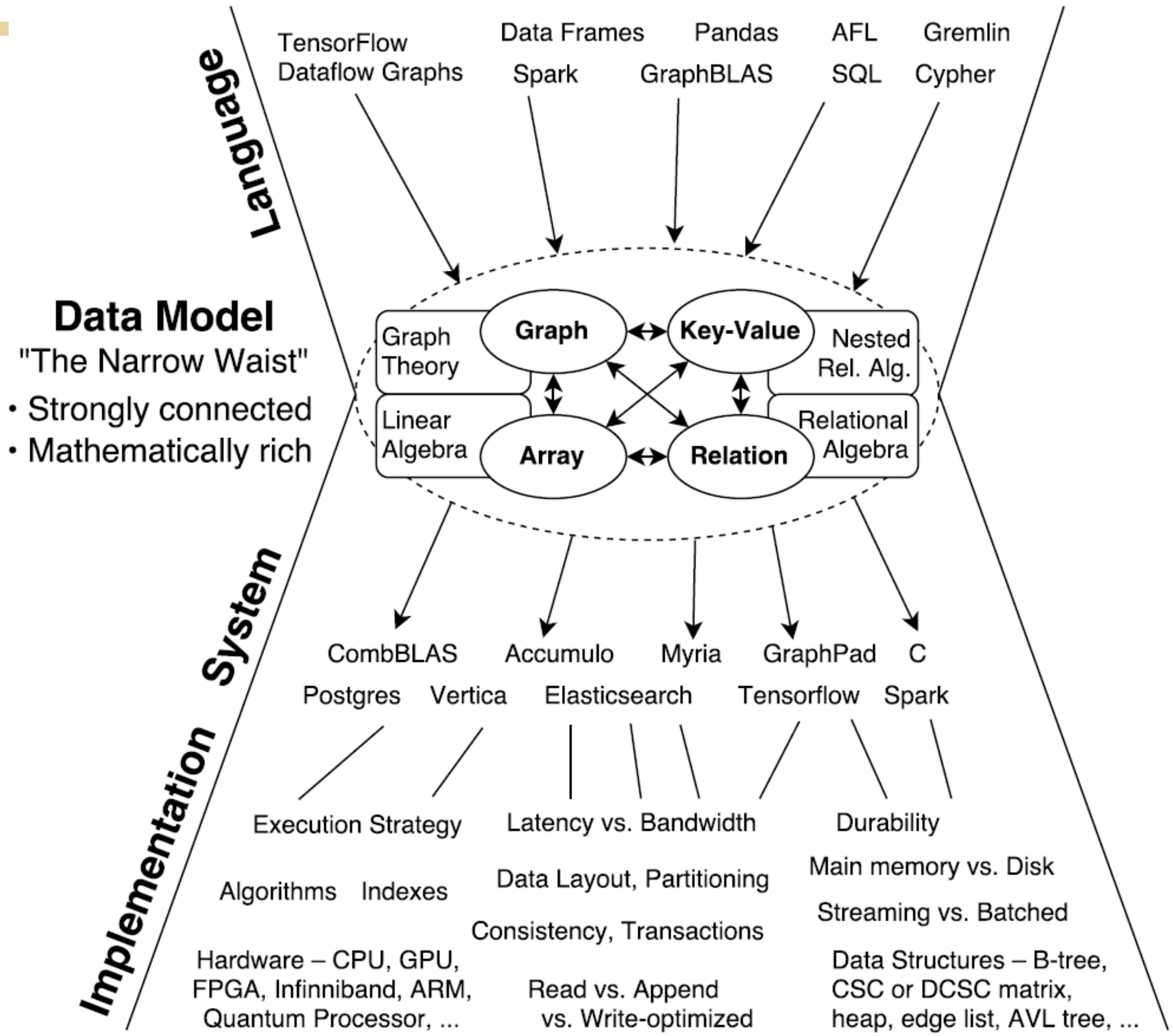
In-Database Analytics for K-V DBs

- > Graphulo: LA on K-V DBs
  - – Graphulo vs. MapReduce experiment
    - > Comparable at scale; in-DB dominates "scale down"
  - – Use in-DB approach for I/O-bound single-pass analytics
- > RACO: RA on K-V DBs
  - – Access Paths helpful as a "physical schema"
  - – Compiles into Accumulo iterators



Future: A Polystore Cross-Platform Optimizer

Why does LA, RA work on K–V DBs?




Some things look better inside of the store.
— Drake —
[redacted]

Backup Slides

Background on Accumulo

<table>
<tr><th colspan="5">Key</th><th rowspan="3"><u>Value</u></th></tr>
<tr><td rowspan="2"><u>Row</u></td><td colspan="3">Column</td><td rowspan="2">Timestamp</td></tr>
<tr><td>Family</td><td><u>Qualifier</u></td><td>Visibility</td></tr>
</table>

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 access control

> Row store by default
  – Scan over rows for O(log n) lookup & sorted order
  – Maintain "Transpose Table" for column-major access path

> Iterator processing framework

MxM: Graphulo vs. MapReduce

Table of Results

SCALE	Entries in Table **C**		Graphulo		MapReduce 10 Reducers	
	PartialProducts	AfterSum	Time (s)	Rate (pp/s)	Time (s)	Rate (pp/s)
10	8.16×10^5	2.69×10^5	8.98×10^{-1}	9.09×10^5	4.95×10^1	1.65×10^4
11	2.34×10^6	7.90×10^5	1.36	1.72×10^6	4.45×10^1	5.26×10^4
12	6.77×10^6	2.41×10^6	2.45	2.77×10^6	4.47×10^1	1.52×10^5
13	1.89×10^7	7.04×10^6	6.74	2.80×10^6	5.15×10^1	3.66×10^5
14	5.32×10^7	2.34×10^7	2.97×10^1	1.79×10^6	6.57×10^1	8.10×10^5
15	1.47×10^8	6.38×10^7	8.21×10^1	1.79×10^6	1.26×10^2	1.17×10^6
16	4.00×10^8	2.30×10^8	2.89×10^2	1.38×10^6	3.44×10^2	1.16×10^6
17	1.09×10^9	9.19×10^8	1.09×10^3	1.00×10^6	9.80×10^2	1.11×10^6
18	2.94×10^9	2.55×10^9	3.67×10^3	7.99×10^5	3.68×10^3	7.97×10^5
19	7.84×10^9	6.58×10^9	1.56×10^4	5.02×10^5	2.23×10^4	3.52×10^5

Past Experiments

Compare Graphulo to

> Single-node in-memory matrix libraries
  - D4M Sparse matrix library (MATLAB)
  - MTJ Dense matrix library (Java)

> Itself
  - Benchmark of Graphulo scalability with cluster size

Tasks:

> Matrix Multiply (MxM), the backbone of LA

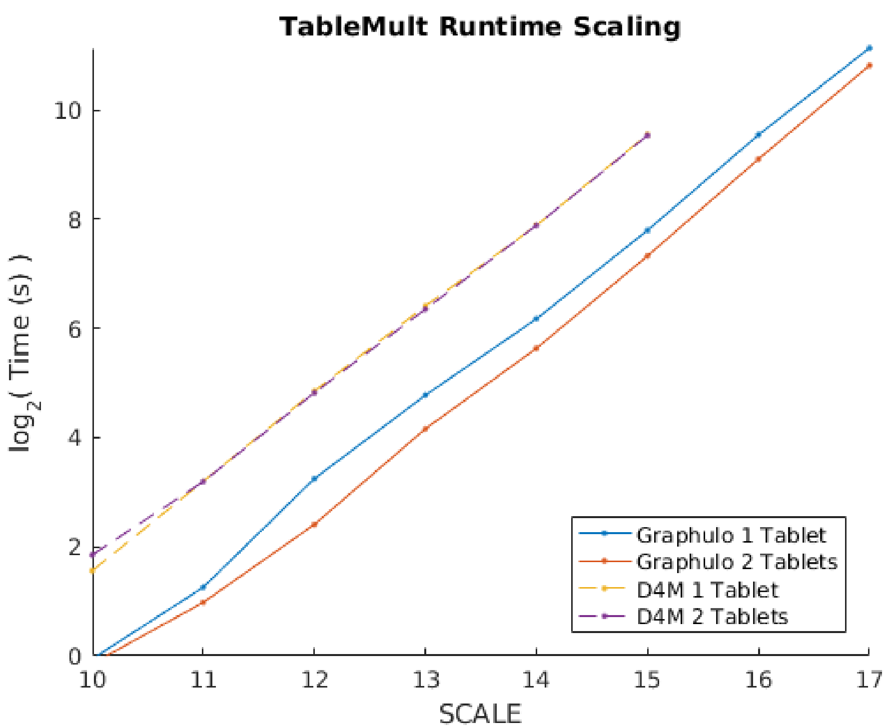
> Subgraph Extraction, via MxM with a cue table

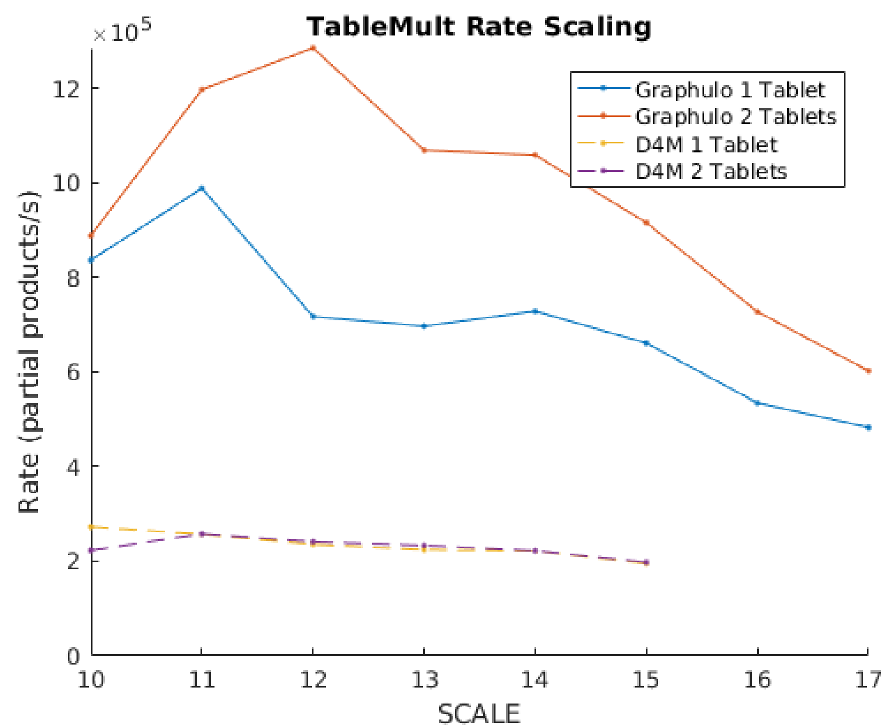
> Jaccard Coefficients, a vertex similarity metric

> k-Truss Subgraph, used to detect communities

- 2015 HPEC: MxM, single-node
  - Graphulo vs. D4M (MATLAB sparse matrix library)
  - Result: Graphulo universally better
- 2016 HPEC: Jaccard, k-Truss, single-node
  - Graphulo vs. D4M vs. MTJ (Java dense matrix library)
  - Result: Graphulo universally better at Jaccard;
D4M/MTJ universally better at k-Truss
- 2016 HPEC: MxM, subgraph extraction, cluster
  - Graphulo vs. itself on increasing cluster size
  - Result: Linear weak & strong scaling on MxM
Subgraph extraction scales, in interactive range

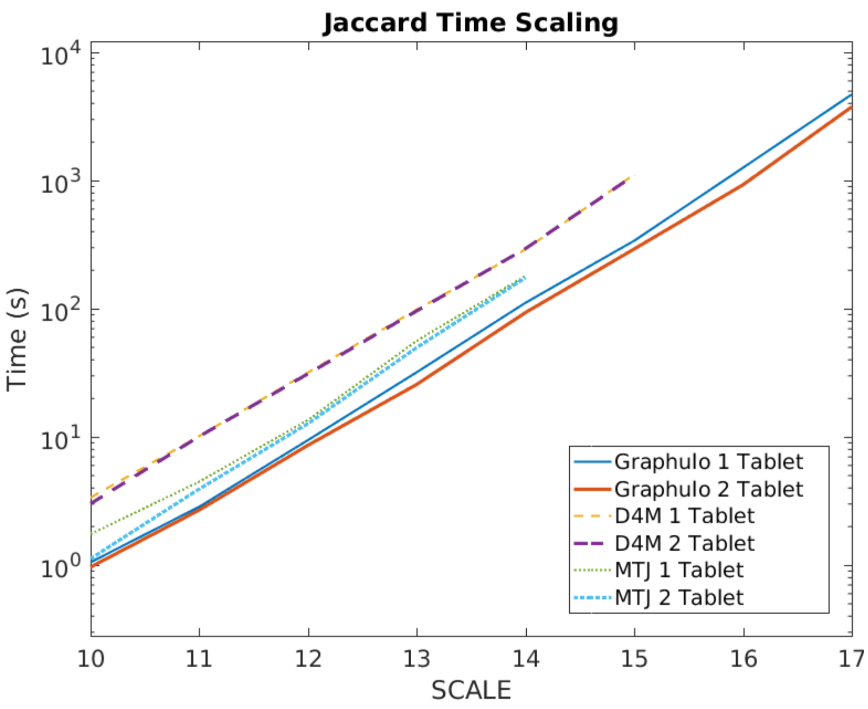
MxM Single-Node Experiment

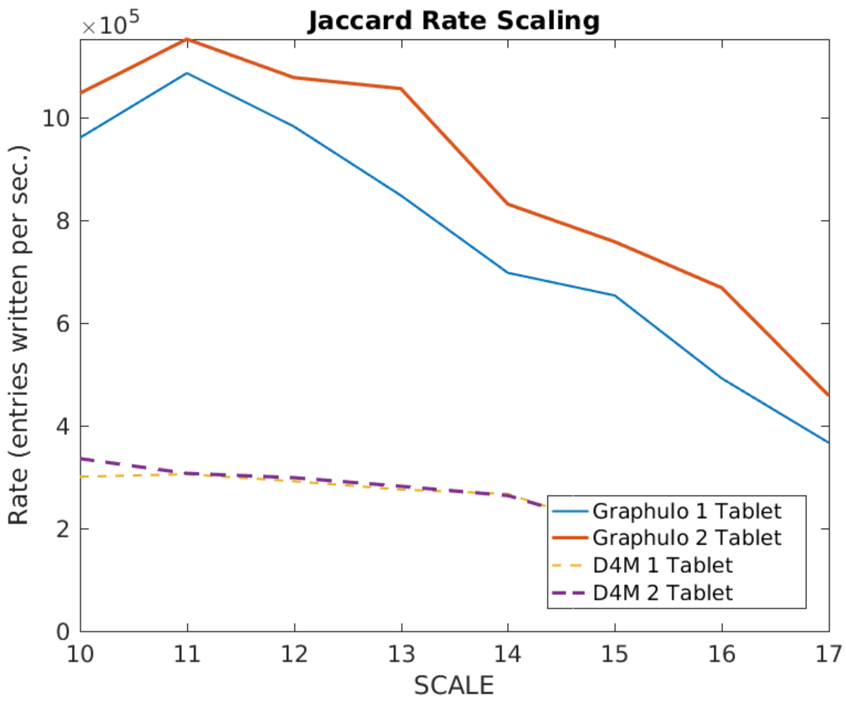




UNIVERSITY *of* WASHINGTON

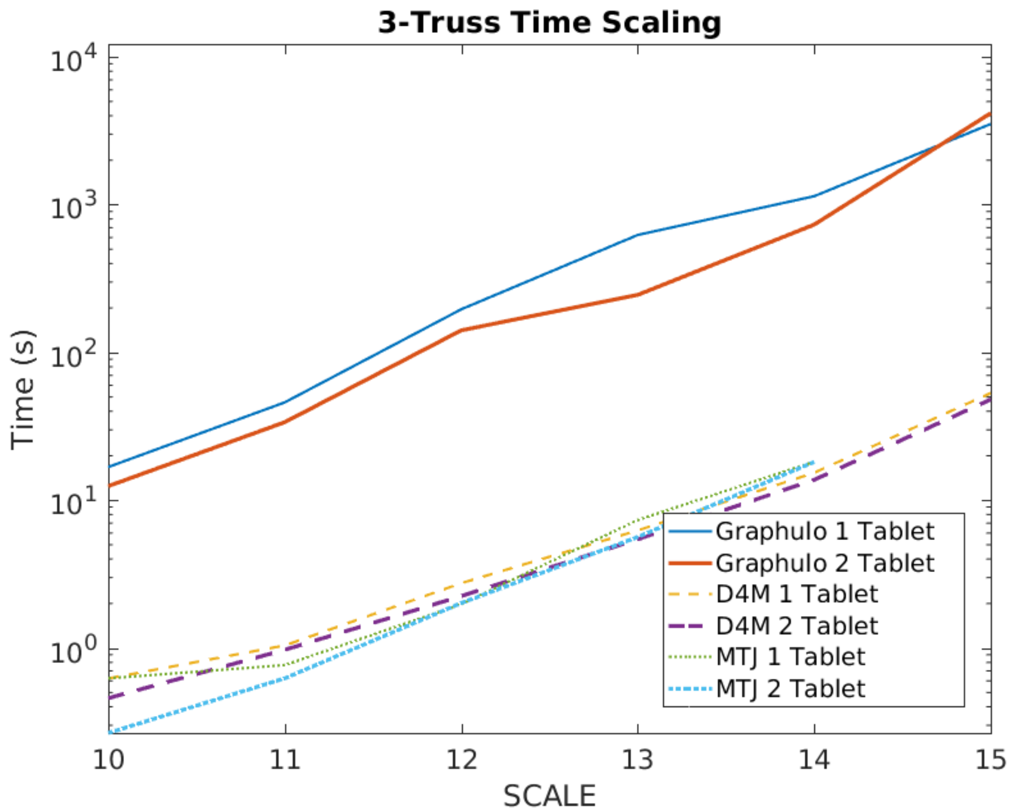
Jaccard Experiment Graphs





UNIVERSITY *of* WASHINGTON

3-Truss Experiment Graph

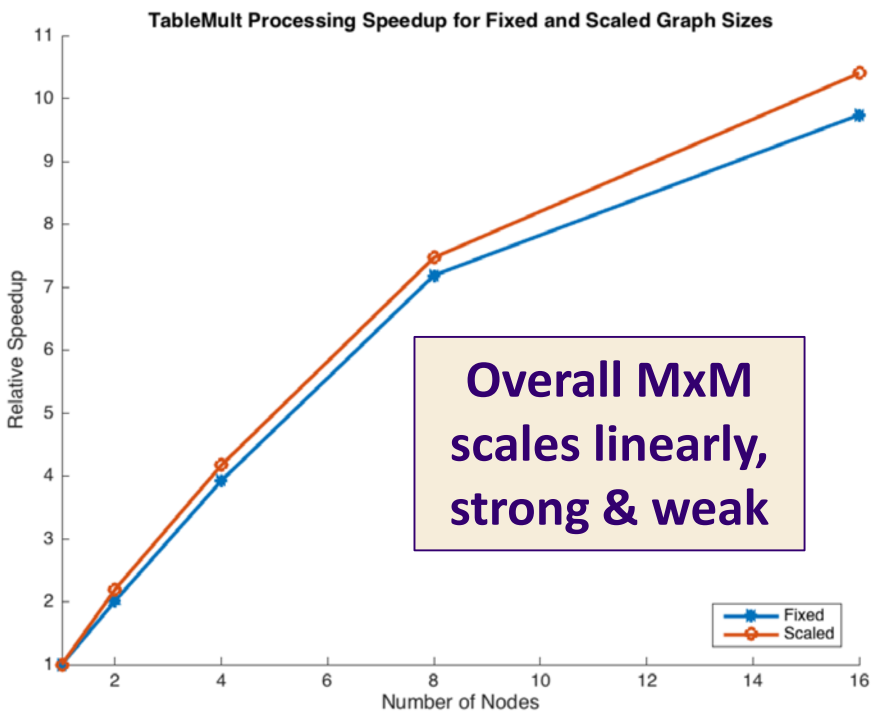


Some points may need re-evaluating

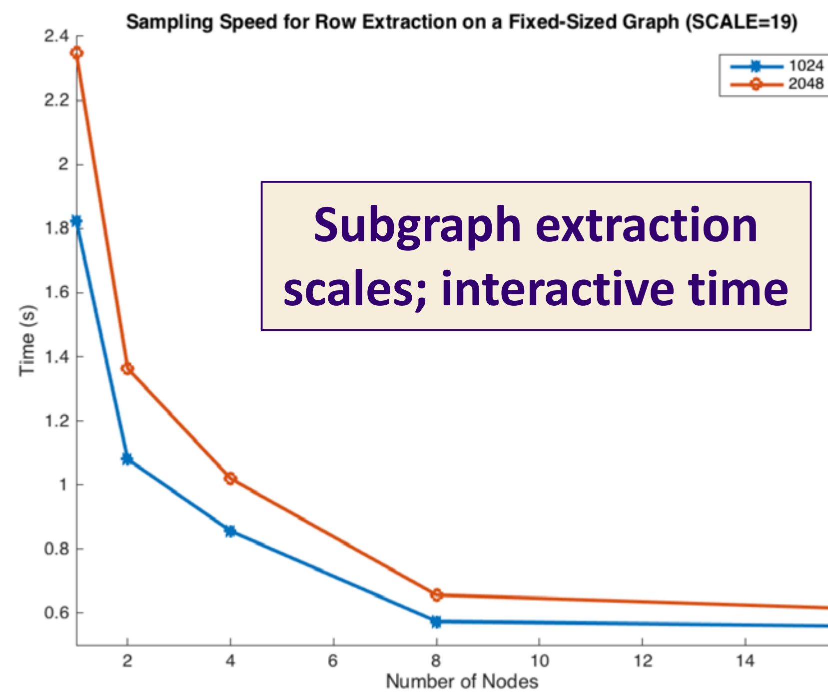
Past Exp: Benchmarking Graphulo

Benchmarking the Graphulo Processing Framework

Timothy Weale[†], Vijay Gadepally[‡§*], Dylan Hutchison[°] and Jeremy Kepner[‡§+]
[†]Department of Defense, [‡]MIT Lincoln Laboratory, [§]MIT Computer Science & AI Laboratory
[+]MIT Mathematics Department, [°]University of Washington

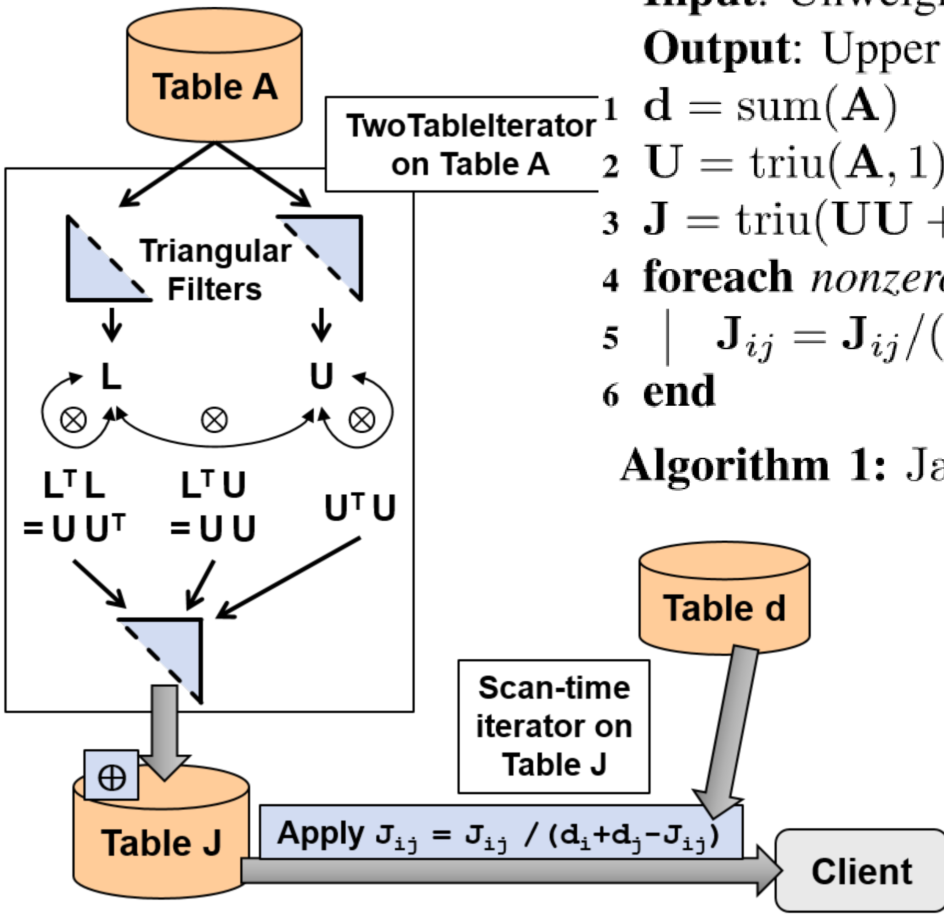


Overall MxM scales linearly, strong & weak



Subgraph extraction scales; interactive time

Alg 1: Jaccard Coefficients



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

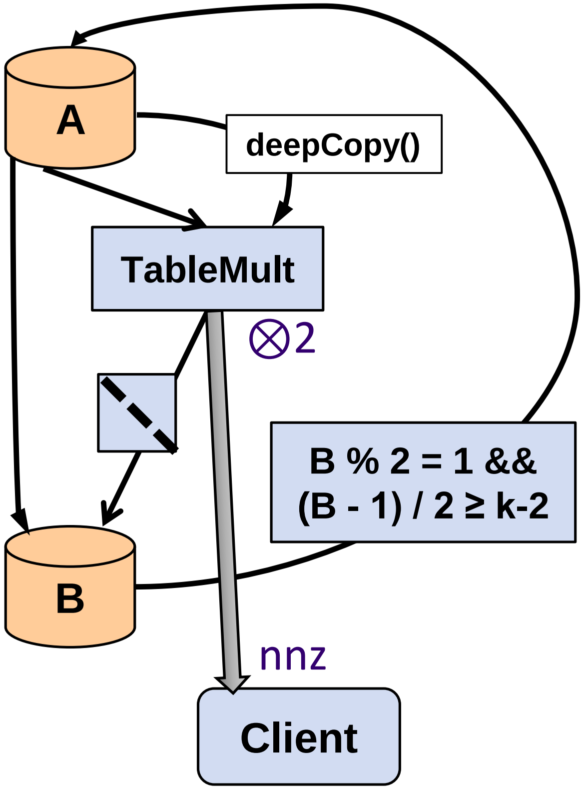
$\mathbf{d} = \mathrm{sum}(\mathbf{A})$ *// pre-computed in degree table*
$\mathbf{U} = \mathrm{triu}(\mathbf{A}, 1)$ *// strict upper triangle filter*
$\mathbf{J} = \mathrm{triu}(\mathbf{UU} + \mathbf{UU}^\intercal + \mathbf{U}^\intercal\mathbf{U}, 1)$ *// fused MxM*
foreach *nonzero entry* $\mathbf{J}_{ij}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

Note! This has been improved since the HPEC paper. Doable as a special OneTable operation.

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*

  $\mathbf{B} = \mathbf{B} + 2\mathbf{AA}$ *// 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