D4M 2.0 Schema: A General Purpose High Performance Schema for the Accumulo Database

Jeremy Kepner, Christian Anderson, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Matthew Hubbell, Peter Michaleas, Julie Mullen, David O’Gwynn, Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee

IEEE HPEC 2013

This work is sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.
Outline

- Introduction
- D4M
- Schema
- Twitter
- Summary
Example Big Data Applications

**ISR**
- Graphs represent entities and relationships detected through multi-INT sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

**Social**
- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

**Cyber**
- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Detect cyber attacks or malicious software

Cross-Mission Challenge: detection of subtle patterns in massive multi-source noisy datasets
**LLSuperCloud Software Stack: Big Data + Big Compute**

- **Novel Analytics for:** Text, Cyber, Bio
- **High Level Composable API:** D4M (“Databases for Matlab”)
- **Distributed Database:** Accumulo (triple store)
- **High Performance Computing:** LLGrid + Hadoop

Combining Big Compute and Big Data enables entirely new domains.
LLSuperCloud Test Bed

- LLSuperCloud allows traditional supercomputing, VMs and Hadoop/Accumulo to dynamically share the same hardware; allows users to:
  - Dynamically stand up and test heterogeneous clouds
  - Integrate different clouds for best mission solution
  - Determine which clouds are best for which mission
Data Storage Landscape

- Leading areas of innovation are in dense structured databases and sparse unstructured databases.

ACID = Atomicity, Consistency, Isolation, Durability
Accumulo “Big Table” Database

- Accumulo is the fastest open source database in the world
- Widely used for gov’t applications
Outline

- Introduction
- D4M
- Schema
- Twitter
- Summary
High Level Language: D4M

http://www.mit.edu/~kepner/D4M

D4M binds associative arrays to databases, enabling rapid prototyping of data-intensive cloud analytics and visualization.

Accumulo Distributed Database

D4M
Dynamic Distributed Dimensional Data Model

Associative Arrays
Numerical Computing Environment

Query: Alice, Bob, Cathy, David, Earl

A D4M query returns a sparse matrix or a graph...

...for statistical signal processing or graph analysis in MATLAB.
D4M Key Concept: 
Associative Arrays Unify Four Abstractions

- Extends associative arrays to 2D and mixed data types
  \[ A('alice','bob') = 'cited' \]
  or \[ A('alice','bob') = 47.0 \]

- Key innovation: 2D is 1-to-1 with triple store
  \('alice','bob','cited'\)
  or \('alice','bob',47.0\)
Composable Associative Arrays

• Key innovation: mathematical closure
  – All associative array operations return associative arrays

• Enables composable mathematical operations
  \[ A + B \quad A - B \quad A \& B \quad A|B \quad A*B \]

• Enables composable query operations via array indexing
  \[ A('alice bob',:) \quad A('alice',:) \quad A('al*',:) \]
  \[ A('alice : bob',:) \quad A(1:2,:) \quad A == 47.0 \]

• Simple to implement in a library (~2000 lines) in programming environments with: 1\textsuperscript{st} class support of 2D arrays, operator overloading, sparse linear algebra

• Complex queries with ~50x less effort than Java/SQL
• Naturally leads to high performance parallel implementation
Database Discovery Workshop

3 day hands-on workshop on:

**Systems**
- Parse, ingest, query, analysis & display

**Usage**
- Files vs. database, chunking & query planning

**Detection theory**
- Clutter, background, detection & tracking

**Technology selection**
- Knowing what to use is as important as knowing how to use it

**Using state-of-the-art technologies:**
- D4M
- Python
- R
- Accumulo
- Hadoop
- SciDB
Outline

- Introduction
- D4M
- Schema
- Twitter
- Summary
Generic D4M Triple Store Exploded Schema

Input Data

<table>
<thead>
<tr>
<th>Time</th>
<th>Col1</th>
<th>Col2</th>
<th>Col3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001-01-01</td>
<td>a</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>2001-01-02</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>2001-01-03</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

Accumulo Table: \( T \)

<table>
<thead>
<tr>
<th></th>
<th>Col1</th>
<th>Col2</th>
<th>Col3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>01-01-2001</th>
<th>02-01-2001</th>
<th>03-01-2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Col2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Col3</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Accumulo Table: \( T^{transpose} \)

- Tabular data expanded to create many type/value columns
- Transpose pairs allows quick look up of either row or column
- Flip time for parallel performance
Tables: SQL vs D4M+Accumulo

SQL Dense Table: T

<table>
<thead>
<tr>
<th>log_id</th>
<th>src_ip</th>
<th>srv_ip</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>128.0.0.1</td>
<td>208.29.69.138</td>
</tr>
<tr>
<td>002</td>
<td>192.168.1.2</td>
<td>157.166.255.18</td>
</tr>
<tr>
<td>003</td>
<td>128.0.0.1</td>
<td>74.125.224.72, 208.29.69.138</td>
</tr>
</tbody>
</table>

Use as row indices

Create columns for each unique type/value pair

Accumulo D4M schema (aka NuWave) Tables: E and Eᵀ

- Both dense and sparse tables stored the same data
- Accumulo D4M schema uses table pairs to index every unique string for fast access to both rows and columns (ideal for graph analysis)
## Queries: SQL vs D4M

<table>
<thead>
<tr>
<th>Query Operation</th>
<th>SQL</th>
<th>D4M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select all</td>
<td><code>SELECT * FROM T</code></td>
<td><code>E(:, :)</code></td>
</tr>
<tr>
<td>Select column</td>
<td><code>SELECT src_ip FROM T</code></td>
<td>`E(:, 'startsWith('src_ip</td>
</tr>
<tr>
<td>Select sub-column</td>
<td><code>SELECT src_ip FROM T WHERE src_ip=128.0.0.1</code></td>
<td>`E(:, 'src_ip</td>
</tr>
<tr>
<td>Select sub-matrix</td>
<td><code>SELECT * FROM T WHERE src_ip=128.0.0.1</code></td>
<td>`E(Row(E(:, 'src_ip</td>
</tr>
</tbody>
</table>

- Queries are easy to represent in both SQL and D4M
- Pedigree (i.e., the source row ID) is always preserved since no information is lost
## Analytics: SQL vs D4M

<table>
<thead>
<tr>
<th>Query Operation</th>
<th>SQL</th>
<th>D4M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram</td>
<td>SELECT COUNT(src_ip) FROM T GROUP BY src_ip</td>
<td>sum(E(:,StartsWith('src_ip</td>
</tr>
<tr>
<td>Graph traversal</td>
<td>SELECT * FROM T WHERE src_ip=128.0.0.1</td>
<td>v0 = 'src_ip</td>
</tr>
<tr>
<td>Graph construction</td>
<td>... many lines ...</td>
<td>A = E(:,StartsWith('src_ip</td>
</tr>
<tr>
<td>Graph eigenvalues</td>
<td>... many lines ...</td>
<td>eigs(Adj(A))</td>
</tr>
</tbody>
</table>

*• Analytics are easy to represent in D4M
• Pedigree (i.e., the source row ID) is usually lost since analytics are a projection of the data and some information is lost*
Outline

• Introduction
• D4M
• Schema
• Twitter
• Summary
Tweets2011 Corpus
http://trec.nist.gov/data/tweets/

- Assembled for Text REtrieval Conference (TREC 2011)*
  - Designed to be a reusable, representative sample of the twittersphere
  - Many languages

- 16,141,812 million tweets sampled during 2011-01-23 to 2011-02-08 (16,951 from before)
  - 11,595,844 undeleted tweets at time of scrape (2012-02-14)
  - 161,735,518 distinct data entries
  - 5,356,842 unique users
  - 3,513,897 unique handles (@)
  - 519,617 unique hashtags (#)


Ben Jabur et al, ACM SAC 2012
Twitter Input Data

<table>
<thead>
<tr>
<th>TweetID</th>
<th>User</th>
<th>Status</th>
<th>Time</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>29002227913850880</td>
<td>Michislipstick</td>
<td>200</td>
<td>Sun Jan 23 02:27:24 +0000 2011</td>
<td>@mi_pegadejeito Tipo. Você ...</td>
</tr>
<tr>
<td>29002228131954688</td>
<td><strong>rosana</strong></td>
<td>200</td>
<td>Sun Jan 23 02:27:24 +0000 2011</td>
<td>para la semana q termino ...</td>
</tr>
<tr>
<td>29002228165509120</td>
<td>doasabo</td>
<td>200</td>
<td>Sun Jan 23 02:27:24 +0000 2011</td>
<td>お腹すいたずえ</td>
</tr>
<tr>
<td>29002228937265152</td>
<td>agusscastillo</td>
<td>200</td>
<td>Sun Jan 23 02:27:24 +0000 2011</td>
<td>A nadie le va a importar ...</td>
</tr>
<tr>
<td>29002229444771841</td>
<td>nob_sin</td>
<td>200</td>
<td>Sun Jan 23 02:27:24 +0000 2011</td>
<td>さて。札幌に帰るか。</td>
</tr>
<tr>
<td>29002230724038657</td>
<td>bimosephano</td>
<td>200</td>
<td>Sun Jan 23 02:27:25 +0000 2011</td>
<td>Wait :)</td>
</tr>
<tr>
<td>29002231177019392</td>
<td>_Word_Play</td>
<td>200</td>
<td>Sun Jan 23 02:27:25 +0000 2011</td>
<td>Shawty is 53% and he pick ...</td>
</tr>
<tr>
<td>29002231202193408</td>
<td>missogeeeb</td>
<td>200</td>
<td>Sun Jan 23 02:27:25 +0000 2011</td>
<td>Lazy sunday (๑´•̀ω•́๑) oooo !</td>
</tr>
<tr>
<td>29002231692922880</td>
<td>PennyCheco06</td>
<td>301</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

- Mixture of structured (TweetID, User, Status, Time) and unstructured (Text)
- Fits well into standard D4M Exploded Schema
### Tweets2011 D4M Schema

#### Accumulo Tables: Tedge/TedgeT

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Tedge/TedgeT</th>
<th>Colum Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>08805831972220092</td>
<td>@mi_pegadejeito</td>
<td>Tipo</td>
</tr>
<tr>
<td>75683042703220092</td>
<td>wait :)</td>
<td></td>
</tr>
<tr>
<td>08822929613220092</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>

#### TedgeDeg

<table>
<thead>
<tr>
<th>Degree</th>
<th>Row Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>838</td>
<td>6</td>
</tr>
<tr>
<td>150</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>327</td>
<td>7</td>
</tr>
<tr>
<td>625</td>
<td>454</td>
</tr>
<tr>
<td>822</td>
<td>596</td>
</tr>
</tbody>
</table>

#### TedgeTxt

<table>
<thead>
<tr>
<th>Row Key</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>08805831972220092</td>
<td>@mi_pegadejeito Tipo. Você fazer uma plaquinha pra mim, com o nome do FC pra você tirar uma foto, pode fazer isso?</td>
</tr>
<tr>
<td>75683042703220092</td>
<td>wait :)</td>
</tr>
<tr>
<td>08822929613220092</td>
<td>null</td>
</tr>
</tbody>
</table>

- **Standard exploded schema indexes every unique string in data set**
- **TedgeDeg** accumulate sums of every unique string
- **TedgeTxt** stores original text for viewing purposes
Tweets2011 (single node database)

- Raw Data: 1.8GB (1621 Files; 100K tweets/file)
  1. Parse: 20 minutes ($N_p = 16$)
  - Triple Files & Assoc Files: 6.7GB (7x1621 files)
  2. Ingest: 40 minutes ($N_p = 3$); 20 entries/tweet, 200K entries/second
    - Accumulo Database: 14GB in 350M entries

- Raw data to fully indexed database < 1 hour
- Database good for queries, assoc files good for complete scans
Accumulo Ingest Performance
Single Node Graph500 Benchmark Data

- Single node high performance ingest requires pre-splitting tables
- Can achieve high parallel ingest on single node with proper splitting
Outline

- Introduction
- Capabilities
- Data
- Analytics
- Summary
Word Tallies

- **D4M Code**
  
  \[ T_{\text{deg}} = \text{DB('TedgeDeg')}; \]
  
  \[ \text{str2num}(T_{\text{deg}}(\text{StartsWith('word|: ')},:) > 20000 \]

- **D4M Result (1.2 seconds, }N_p = 1\)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>:</td>
</tr>
<tr>
<td>word</td>
<td>:</td>
</tr>
<tr>
<td>word</td>
<td>:</td>
</tr>
<tr>
<td>word</td>
<td>:D</td>
</tr>
<tr>
<td>word</td>
<td>:P</td>
</tr>
<tr>
<td>word</td>
<td>:p</td>
</tr>
</tbody>
</table>

- **Sum table** \( T_{\text{edgeDeg}} \) allows tallies to be seen instantly
Users Who ReTweet the Most
Problem Size

• **D4M Code to check size of status codes**
  
  \[T_{\text{deg}} = \text{DB('TedgeDeg')}\; ;\; T_{\text{deg}}(\text{StartsWith('stat|')})\]

• **D4M Results (0.02 seconds, \(N_p = 1\))**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Count</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat</td>
<td>200</td>
<td>10864273</td>
</tr>
<tr>
<td>stat</td>
<td>301</td>
<td>2861507</td>
</tr>
<tr>
<td>stat</td>
<td>302</td>
<td>836327</td>
</tr>
<tr>
<td>stat</td>
<td>403</td>
<td>825822</td>
</tr>
<tr>
<td>stat</td>
<td>404</td>
<td>753882</td>
</tr>
<tr>
<td>stat</td>
<td>408</td>
<td>1</td>
</tr>
</tbody>
</table>

• **Sum table** \(T_{\text{edgeDeg}}\) indicates 836K retweets (~5% of total)
• **Small enough to hold all TweetIDs in memory**
• **On boundary between database query and complete file scan**
Users Who ReTweet the Most
Parallel Database Query

• **D4M Parallel Database Code**

\[ T = DB('Tedge','TedgeT'); \]
\[ Ar = T(:, 'stat|302 '); \]
\[ my = \text{global\_ind(zeros(size(Ar,2),1,\text{map([Np 1],{},0:Np-1)}));} \]
\[ An = \text{Assoc('', '', '');} \quad N = 10000; \]
\[ \text{for } i=\text{my(1):N:my(end)} \]
\[ \quad Ai = \text{dblLogi(T(Row(Ar(i:min(i+N,my(end)),:),:)),:));} \]
\[ \quad An = An + \text{sum(Ai(:,startsWith('user|',))),1);} \]
\[ \text{end} \]
\[ Asum = \text{gagg(Asum > 2);} \]

• **D4M Result (130 seconds, }N_p = 8\)**

user|Puque007 103, user|Say 113,
user|carp_fans 115, user|habu_bot 111,
user|kakusan_RT 135, user|umaitofu 116

• Each processor queries all the retweet TweetIDs and picks a subset
• Processors each sum all users in their tweets and then aggregate
Users Who ReTweet the Most
Parallel File Scan

- **D4M Parallel File Scan Code**

  ```
  Nfile = size(fileList);
  my = global_ind(zeros(Nfile,1,map([Np 1],{},0:Np-1)));
  An = Assoc('','',);
  for i=my
    load(fileList{i});
    An = An + sum(A( Row(A(:,'
stat|302,')),StartsWith('user|,')),1);
  end
  An = gagg(An > 2);
  ```

- **D4M Result (150 seconds, Np = 16)**

  user|Puque007    100,    user| utilisateurSay 113,
  user|carp_fans    114,    user|habu_bot      109,
  user|kakusan_RT    135,    user|umaitofu      114

- Each processor picks a subset of files and scans them for retweets
- Processors each sum all users in their tweets and then aggregate
Summary

• Big data is found across a wide range of areas
  – Document analysis
  – Computer network analysis
  – DNA Sequencing

• Non-traditional, relaxed consistency, triple store databases are the backbone of many web companies

• D4M Schema provides a general, high performance approach for harnessing the power of these databases