Signal Processing on Databases

Jeremy Kepner

Lecture 3: Entity Analysis in Unstructured Data

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
  – Webolution
  – As is, is OK
  – D4M

• Technologies

• Results

• Demo

• Summary
Primordial Web
Kepner & Beaudry 1992, Visual Intelligence Corp (now GE Intelligent Platforms)

- Browser (html): http put ←→ http get
- Server (http): Gopher
- Database (sql): SQL data

Language:
- perl

Client

Server

Database

- Browser GUI? HTTP for files? Perl for analysis? SQL for data?
- A lot of work just to view data.
- Won’t catch on.
Browser (html):          Server (http):          Database (sql):

http put                 SQL                  http get

http get                 data

Browser GUI? HTTP for files? Perl for analysis? SQL for data?
A lot of work to view a little data.
Won’t catch on.
Modern Web

Game (data):
- http put
- http get

Server (http):
- java
- data

Database (triples):
- BigTable
- riak
- Cassandra
- HBase

Language:
- Java
- PHP
- perl
- Python
- Ruby

Client

Server

Database

- Game GUI! HTTP for files? Perl for analysis? Triples for data!
- A lot of work to view a lot of data.
- Great view. Massive data.
## Modern Web

### Game (data):
- [Image of game data]

### Server (http): Apache Windows Server
- [Images of Apache and Windows Server]
- [Diagram showing http put and http get]
- [Language: Java, PHP, perl, python, Ruby]

### Database (triples):
- [Images of BigTable, riak, Cassandra, HBase]

### Client
- [Images of mobile device and laptop]

### Server

### Database

| • Game GUI! HTTP for files? Perl for analysis? Triples for data! |
| • A lot of work to view a lot of data. Missing middle. |
| • Great view. Massive data. |

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see [http://ocw.mit.edu/help/faq-fair-use/](http://ocw.mit.edu/help/faq-fair-use/).
Future Web?

Game (data):

Server (files):

Database (triples):

Language:

- Game GUI! Fileserver for files! D4M for analysis! Triples for data!
- A little work to view a lot of data. Securely.
- Great view. Massive data.
D4M: “Databases for Matlab”

- D4M binds Associative Arrays to Triple Store, enabling rapid prototyping of data-intensive cloud analytics and visualization.
Outline

- Introduction
- Technologies
  - Hardware
  - Cloud software
  - Associative Arrays
- Results
- Demo
- Summary
What is LL Grid?

- LLGrid is a ~500 user ~2000 processor system
- World’s only desktop interactive supercomputer
  - Dramatically easier to use than any other supercomputer
  - Highest fraction of staff using (20%) supercomputing of any organization on the planet
- Foundation of Supercomputing in Massachusetts
Why is LLGrid easier to use?

Universal Parallel Matlab programming

```matlab
Amap = map([Np 1],{},0:Np-1);
Bmap = map([1 Np],{},0:Np-1);
A = rand(M,N,Amap);
B = zeros(M,N,Bmap);
B(:, :) = fft(A);
```

- pMatlab runs in all parallel Matlab environments
- Only a few functions are needed
  - Np
  - Pid
  - map
  - local
  - put_local
  - global_index
  - agg
  - SendMsg/RecvMsg

- Distributed arrays have been recognized as the easiest way to program a parallel computers since the 1970s
  - Only a small number of distributed array functions are necessary to write nearly all parallel programs
- LLGrid is the first system to deploy interactive distributed arrays
# Cloud Computing Concepts

## Data Intensive Computing
- Compute architecture for large scale data analysis
  - Billions of records/day, trillions of stored records, petabytes of storage
    - Google File System 2003
    - Google MapReduce 2004
    - Google BigTable 2006
- Design Parameters
  - Performance and scale
  - Optimized for ingest, query and analysis
  - Co-mingled data
  - Relaxed data model
  - Simplified programming
- Community:

## Utility Computing
- Compute services for outsourcing IT
  - Concurrent, independent users operating across millions of records and terabytes of data
    - IT as a Service
    - Infrastructure as a Service (IaaS)
    - Platform as a Service (PaaS)
    - Software as a Service (SaaS)
- Design Parameters
  - Isolation of user data and computation
  - Portability of data with applications
  - Hosting traditional applications
  - Lower cost of ownership
  - Capacity on demand
- Community:
The Big Four Cloud Ecosystems

- **Enterprise**
  - IaaS
    - Interactive
    - On-demand
    - Elastic

- **Supercomputing**
  - PaaS
    - High performance
    - Parallel Languages
    - Scientific computing

- **Big Data**
  - PaaS
    - Java
    - Map/Reduce
    - Easy admin

- **DBMS**
  - SaaS
    - Indexing
    - Search
    - Security

- Each ecosystem is at the center of a multi-$B market
- Pros/cons of each are numerous; diverging hardware/software
- Some missions can exist wholly in one ecosystem; some can’t
• LLGrid provides interactive, on-demand supercomputing
• Accumulo database provides high performance indexing, search, and authorizations within a Hadoop environment
**The Big Four Cloud Ecosystems**

- **IaaS**
  - Interactive
  - On-demand
  - Elastic

- **PaaS**
  - High performance
  - Parallel Languages
  - Scientific computing

- **SaaS**
  - Indexing
  - Search
  - Security

- **MapReduce**

- **LLGrid**

- **Supercomputing**

- **Big Data**

- **DBMS**

- **D4M**

- **MapReduce** provides map/reduce interface to supercomputing
- **D4M** provides an interactive parallel scientific computing environment to databases
Big Compute + Big Data Stack

Novel Analytics for: Text, Cyber, Bio

High Level Composable API: D4M (“Databases for Matlab”)

Distributed Database: Accumulo/HBase (triple store)

High Performance Computing: LLGrid + Hadoop

- Combining Big Compute and Big Data enables entirely new domains
Hadoop: Strengths and Weaknesses

• What works well
  – Distributed processing of large data
    Indexing log files
    Sorting data
  – Scale up from single servers to thousands of machines
    Local computation and storage
  – Detect and handle failures at the application layer
    Highly-available service on top of a cluster of computers

• Some difficulties are
  – Controlling compute resources for a given job
    Full blown, greedy scheduling
  – Multi-user environments
    Not easy to provide fair-share control on their use of Hadoop cluster
  – Non-Java programmers
    Takes time to learn the parallel programming API for Java
LLGrid_MapReduce provides a language agnostic and scheduler agnostic map/reduce interface in a supercomputing environment.
• Introduction

• Technologies
  – Hardware
  – Cloud software
  – D4M

• Results

• Demo

• Summary
Multi-Dimensional Associative Arrays

- Extends associative arrays to 2D and mixed data types
  
  $A(\text{\textquoteleft alice \textquoteright \textquoteright}, \text{\textquoteleft bob \textquoteright \textquoteright}) = \text{\textquoteleft cited \textquoteright \textquoteright}$

  or

  $A(\text{\textquoteleft alice \textquoteright \textquoteright}, \text{\textquoteleft bob \textquoteright \textquoteright}) = 47.0$

- Key innovation: 2D is 1-to-1 with triple store
  
  (\text{\textquoteleft alice \textquoteright \textquoteright}, \text{\textquoteleft bob \textquoteright \textquoteright}, \text{\textquoteleft cited \textquoteright \textquoteright})

  or

  (\text{\textquoteleft alice \textquoteright \textquoteright}, \text{\textquoteleft bob \textquoteright \textquoteright}, 47.0)
Composable Associative Arrays

- Key innovation: mathematical closure
  - all associative array operations return associative arrays

- Enables composable mathematical operations

\[
\begin{align*}
    A + B & \quad A - B \quad A \& B \quad A | B \quad A \ast B
\end{align*}
\]

- Enables composable query operations via array indexing

\[
\begin{align*}
    A(\text{`alice bob '},:) & \quad A(\text{`alice '},:) & \quad A(\text{`al* '},:)
    \\
    A(\text{`alice : bob '},:) & \quad A(1:2,:) & \quad A == 47.0
\end{align*}
\]

- Simple to implement in a library (~2000 lines) in programming environments with: 1st 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
Universal “Exploded” Schema

Input Data

<table>
<thead>
<tr>
<th>Time</th>
<th>src_ip</th>
<th>domain</th>
<th>dest_ip</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001-01-01</td>
<td>a</td>
<td>a</td>
<td></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>

Triple Store Table: Ttranspose

<table>
<thead>
<tr>
<th></th>
<th>2001-01-01</th>
<th>2001-01-02</th>
<th>2001-01-03</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_ip/a</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>src_ip/b</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>domain/b</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>domain/c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest_ip/a</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>dest_ip/c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Triple Store Table: T

<table>
<thead>
<tr>
<th></th>
<th>src_ip/a</th>
<th>src_ip/b</th>
<th>domain/b</th>
<th>domain/c</th>
<th>dest_ip/a</th>
<th>dest_ip/c</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001-01-01</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2001-01-02</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001-01-03</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Key Innovations

- Handles all data into a single table representation
- Transpose pairs allows quick look up of either row or column
Outline

- Introduction
- Technologies
- Results
  - Benchmark performance
  - Facet search
  - Management and monitoring
- Demo
- Summary
Stats Diagram

**Triple Store Table: T**

<table>
<thead>
<tr>
<th>Row</th>
<th>Key (time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2001-10-01 01 01 00</td>
</tr>
<tr>
<td>2</td>
<td>2001-10-01 01 02 00</td>
</tr>
<tr>
<td>3</td>
<td>2001-10-01 01 03 00</td>
</tr>
<tr>
<td>4</td>
<td>2001-10-01 01 04 00</td>
</tr>
<tr>
<td>5</td>
<td>2001-10-01 01 05 00</td>
</tr>
<tr>
<td>6</td>
<td>2001-10-01 01 06 00</td>
</tr>
</tbody>
</table>

**Associative Array: A**

- Copy a set of rows from T into associative array A
- Perform the following statistical calculations on A
  - Column count: how many times each column appears in A
  - Column type count: how many times each column type appears in A
  - Column covariance: how many times a each pair of columns in A appear in the same row together
  - Column covariance: how many times a each pair of column types in A appear in the same row together

- Good for identifying column types, gaps, clutter, and correlations
**Stats Diagram**

### Triple Store Table: T

<table>
<thead>
<tr>
<th>Row</th>
<th>Key (time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2001-10-01 01 01 00</td>
</tr>
<tr>
<td>2</td>
<td>2001-10-01 01 02 00</td>
</tr>
<tr>
<td>3</td>
<td>2001-10-01 01 03 00</td>
</tr>
<tr>
<td>4</td>
<td>2001-10-01 01 04 00</td>
</tr>
<tr>
<td>5</td>
<td>2001-10-01 01 05 00</td>
</tr>
<tr>
<td>6</td>
<td>2001-10-01 01 06 00</td>
</tr>
</tbody>
</table>

**Associative Array: A**

- Copy a set of rows from T into associative array A
- Perform the following statistical calculations on A
  - Column count: how many times each column appears in A
  - Column type count: how many times each column type appears in A
  - Column covariance: how many times a each pair of columns in A appear in the same row together
  - Column covariance: how many times a each pair of column types in A appear in the same row together

- Good for identifying column types, gaps, clutter, and correlations
Stats Implementation

- Define a set of rows

\[ r = '2001-01-01 01 02 00,2001-01-01 01 03 00, 2001-01-01 01 04 00, \]

- Copy rows from table to associative array and convert '1' to 1

\[ A = \text{double} (\text{logical} (T(r,:))) \]
\[ A = A(:, 'src_ip/ *,domain/ *,dest_ip/ *,') \]

- Find popular columns counts

\[ \text{sum}(A,1) > 200 \]

- Find popular pairs

\[ A' \ast A > 200 \quad \text{or} \quad \text{sqIn}(A) > 200 \]

- Find domains with many dest IPs

\[ \text{sum} (\text{double} (\text{logical} (\text{sqIn}(A))), 2) > 3 \]
• Very easy to get elementary count info necessary for finding clutter and anomalies
Adjacency matrix a natural result of covariance calculation
Facet Search

• Core analytic of SKS

• Give keyword distribution of a set of documents that share a common keyword(s)
  – Provides useful guide to what keyword to select next

• Currently implemented with several hundreds of lines of Java/SQL

• Associative array implementation has 1 line
## Facet Search Algorithm

- Associative array relates documents to place, org and person entities

\[ A(x,y) : S^{N \times M} \rightarrow R \]

- Facets \( y_1 = \text{UN}, \ y_2 = \text{Carl} \)

- Documents that contain both

\[ A(:,y_1) \& A(:,y_2) \]

- Entity counts in the above set of documents obtained via matrix multiply

\[ (A(:,y_1) \& A(:,y_2))^t A \]
Outline

• Introduction
• Technologies
• Results
• Demo
• Summary
Reuters Corpus V1 (NIST)
1996-08-20 to 1997-08-19 (Released 2000-11-03)

- 810,000 Reuters news blurbs
- Picked 70,000 and found 13,000 entities
- A is a 70Kx13K associative array with 500K entries

- Power laws everywhere
- Number of persons >> number of places
- Number of document/places >> number of document/person
% Connecting to a database.
• Web evolution has resulted in a new class of technologies for
  – Display (game interfaces)
  – Analysis (D4M)
  – Storage (triple stores)
• D4M is a novel technology that allows complex analytics to be
  implement with significantly less effort than traditional
  approaches
• D4M is built on composable associative arrays which admit linear
  algebraic manipulation
Example Code & Assignment

• Example Code (end of Lecture 3 and start of lecture 4)
  – d4m_api/examples/2Apps/1EntityAnalysis
  – d4m_api/examples/2Apps/2TrackAnalysis

• Assignment
  – None