MAD Skills: New Analysis Practices for Big Data

Presented By:
Christian Grant
cgrant@cise.ufl.edu
MAD Skills: New Analysis Practices for Big Data

• Authors
  • Jeff Cohen – Greenplum
  • Brian Dolan – Fox Audience Network
  • Mark Dunlap – Evergreen Technologies
  • Joseph M Hellerstein – UC Berkeley
  • Caleb Welton – Greenplum

• Presented at Very Large Database Conference 2009 in Lyon, France
If you are looking for a career where your services will be in high demand, you should find something where you provide a scarce, complementary service to something that is getting ubiquitous and cheap.

So what’s getting ubiquitous and cheap? Data.

And what is complementary to data? Analysis.

– Prof. Hal Varian, Chief Economist at Google
• Storage is Cheap

• The world’s largest data warehouse of ~15 years ago can be stored on disks for about $2000.
Traditionally, a lot of time is spent building well structured data warehouses.

- Contains summaries of data
- Main analytics location
- Jealously guarded by IT
• Data analysis is common culture

• Culture is to collect and analyze data in different business units.
• Data analysis is common culture

• Culture is to collect and analyze data in difference business units.

• “In God we trust; all others must bring data”
• Data analysis is common culture

• Culture is to collect and analyze data in difference business units.

• “In God we trust; all others must bring data”

• Analytics are crucial – Analysts need more access and more permissions

• Hence M.A.D.
• Magnetic – Attract data and practitioners. The Database must be painless and efficient.

• Agile – Analysts need to ingest and analyze on the fly.

• Deep – Sophisticated analytics at scale. Don’t force analyst to work on samples (where the long tail is lost).

• Library of tools
Outline

• Background
• FOX Audience Network
• Being MAD
• In-Database Operations
• MAD DBMS
• Conclusions
• Question
Background

- OLAP and Data Cubes
  - Data structure to provides descriptive statistics – **Simple** summaries (sum, average, std)
  - We want inferential/inductive statistics – Predictions, causality analysis, distributional comparison
Background

• Databases and Statistical Packages
  • Many analysts download data to use in Excel/SAS/Matlab/R or their favorite programming language? FORTRAN??
  • Use matrix/vector operations
  • Most of these stat packages require data to fit in RAM
    • Taking samples from the full data to fit into ram results in loss of precision
  • External toolkits may also lack parallelism
Background

• MapReduce and Parallel Programming
  • Strengths of MR are scalable batch parallelism and fault tolerance.
  • There is work on putting ML algorithms in MapReduce Framework (Mahout)
  • MR is just a programming framework and works well with MAD
Background

• Data Mining and Analytics in the DB
  • Plenty of work in this area
  • Methods are typically implemented as a black box
  • Algorithms are not flexible as in programmed packages
FOX Audience Network

- Served MySpace.com, IGN.com, Scout.com etc.
- About 150 Million users
- Ad network, bought by Rubicon in Nov 2010
Fox Audience Network

- 42 Nodes (2 Masters, 40 Workers)
- Sun X4500 Thumper
- 48 500GB drives
- 16GB RAM
- ~ 5TB Daily
- 1 Table 1.5 Trillion Rows
- Many different types of users workloads
- Dynamic query ecosystem
Fox Audience Network

• Use Cases
• Queries to the system are vastly different
• Example queries:
  • *How many female WWF enthusiasts under the age of 30 visited the Toyota community over the last four days and saw a medium rectangle?*
  • *Expensive*
  • *How are these people similar to those that visited Nissan?*
  • Open-ended, requires some statistics and the analyst to be in the loop.
Being MAD

• Analysts (Data Scientists) trump DBAs
  • They crave data
  • They can work with dirty data
  • They like all data (not samples)
  • Have a belt of tools
Being MAD

- Get data to the warehouse as soon as possible
- Cleaning and integration should be done intelligently
Being MAD

• Three **logical** layers for data stores:
  • Reporting – Specialized static aggregates
  • Production – Aggregates for most users
  • Stages – Raw tables and logs
  • *Sandboxes* – Play ground for analysts
• The logical levels encourages creativity
Being MAD

• Data Scientists need power tools Math + Stats + ML

• The MADLib approach is to develop a *hierarchy* of mathematical concepts in the database.
In-Database Operations

- User Defined Functions (UDF)
- SQL or PL/pgSQL
- Internal (c)
- Dynamically Loaded (c, c++, java, javascript, perl, python, R, ruby, scheme, etc)

- **SELECT** \( f_1 \)(city) **FROM** states **WHERE** \( f_2 \)(capital)
In-Database Operations

- Value abstraction levels
  - 1.0
  - [0.3, 0.3, 0.4]
  - [[0.3, 0.7], [0.6, 0.4]]
  - $f(\cdot) \rightarrow \text{value}$
In-Database Operations

Scalar Arithmetic
In-Database Operations

- Scalar Arithmetic
  - `SELECT 5*4;`
  - `SELECT sqrt(64);`
  - `SELECT cos(-3.14159 * sqrt(2) / 2 );`
In-Database Operations

Vector Arithmetic
In-Database Operations

- Vector
  - array objects: float8[
  - array operations are implemented as UDFs.

Technically violates 1st normal form
In-Database Operations

- Vector

```sql
CREATE OR REPLACE FUNCTION MADLIB_SCHEMA.array_add(x anyarray, y anyarray) RETURNS anyarray
AS 'MODULE_PATHNAME', 'array_add'
LANGUAGE C IMMUTABLE;

CREATE OPERATOR + ( 
    leftarg = numeric[], 
    rightarg = numeric[], 
    procedure = array_add, 
    communicator = +
);
```

technically violates 1st normal form
In-Database Operations

Matrix Arithmetic
In-Database Operations

• Matrix Representation #1

```sql
CREATE TABLE B (
row_number integer,
vector numeric[]
);
```

```sql
INSERT INTO B (vector) VALUES ('{8,8}');
INSERT INTO B (vector) VALUES ('{2,2}');
```

http://www.postgresql.org/docs/current/static/arrays.html
In-Database Operations

- Matrix addition:

  ```sql
  SELECT A.row_number, A.vector + B.vector
  FROM A, B
  WHERE A.row_number = B.row_number
  ```

  This can be easily implemented as a primitive operator:

  ```sql
  SELECT A ** B from A ,B
  ```
In-Database Operations

- Matrix and vector multiplication: $A\mathbf{v}$

```sql
SELECT 1, array_accum(row_number, vector*\mathbf{v})
FROM A
```

`array_accum(x,\mathbf{v})` is a custom function. It takes the value $\mathbf{v}$ and puts it in the row indexed by $x$. 
In-Database Operations

- Matrix Representation #2

```sql
CREATE TABLE A (  
row_number INTEGER,  
column_number INTEGER,  
value DECIMAL  
);
```

- Great sparse representation!

http://www.postgresql.org/docs/current/static/arrays.html
In-Database Operations

• Matrix Multiplication (using rep #2):

```
SELECT A.row_number, B.column_number, SUM(A.value * B.value)
FROM A, B
WHERE A.column_number = B.row_number
GROUP BY A.row_number, B.column_number
```
In-Database Operations

• These operations require one pass over the data and can be done in parallel.

• What about Matrix Division, and Matrix Inverse?

• SQL is awkward bc of it’s lack of loops for iteration.
In-Database Operations

• Document similarity for fraud detection
  • Check to see if several advertisers are linking to pages to the same content.
  • If several advertisers have out links to similar documents it is possible they are using stolen credit card.
  • Especially if the outbound documents are malicious.
In-Database Operations

• For document similarity we use \textit{tf-idf}. This is a score for a term in a given document.

For a term:

\[ \text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \]

\[ \text{tf}_{t,d} = \text{term count in doc} \]

\[ \text{idf}_t = \log \left\{ \frac{|\text{docs}|}{|\text{docs w. term}|} \right\} \]
In-Database Operations

- Table $docs$ contains triples $(document, term, count)$

- **tf** UDF in SQL takes a term and document as parameters
  
  ```sql
  SELECT count(term)
  FROM docs
  WHERE term = t AND document = d
  ```

- **idf** UDF in SQL takes a term as a parameter
  
  ```sql
  SELECT log ( 
    (SELECT count(document) FROM docs) / 
    (SELECT count(document) FROM docs WHERE term = t GROUP BY document))
  FROM docs
  ```
In-Database Operations

• Each doc will have a vector with the tf-idf scores, we can compare the results using cosine similarity.

\[ \cos \theta = \frac{v \cdot u}{||v|| \cdot ||u||} \]

small \( \theta \) means \( v \) and \( u \) are similar

• SQL is straightforward using the vector operators
Ordinary Least Squares is used to fit a line to a collection of points.

This helps to describe seasonal trends - it's a first look at the data.

Solve $Y = X\beta$ where $X$ is an $n \times k$ Matrix.

Estimate $\beta$ using $\beta^* = (X^TX)^{-1}X^Ty$

We can parallelize using computation

Distribute: $A \leftarrow X^TX, \ b \leftarrow X^Ty$

Collect: $A$ and $b$.

Calculate: $A^{-1}$

Multiply: $A^{-1} * b$
In-Database Operations

Functional Arithmetic
In-Database Operations

• A/B Testing (Multivariate testing)
  • Web companies show people different versions of their website to see which one is the best.
  • Ad companies compare different add campaigns and may want to find the one with the higher clicks-through rate.
In-Database Operations

• Mann-Whitney U Test

  • This is a popular test to compare non-parametric data.

  • non-parametric def= data set that does not fit in a well known distribution

  • Combine data values and score the values, find the average location of the list for each value.

  • $Z = U - \left(\frac{n_1n_2}{2}\right) / \sqrt{\left(\frac{n_1n_2(n_1 + n_2 + 1)}{12}\right)}$
In-Database Operations

- Mann-Whitney U Test
  - Given Table $T$ with $(sample_id, value)$

```sql
CREATE VIEW $R$ AS
SELECT sample_id, avg(value) AS sample_avg,
  sum(rown) AS rank_sum, count(*) AS sample_n,
  sum(rown) - count(*) * count(*) + 1 AS sample_us
FROM ( SELECT sample_id,
  row_number() OVER (ORDER BY value DESC) AS rown, value FROM $T$ ) AS ordered
GROUP BY sample_id
```
In-Database Operations

- Mann-Whitney U Test
  - With large sample count we can get the z_score

```sql
SELECT r.sample_u, r.sample_avg, r.sample_n,
      ((r.sample_u - a.sum_u / 2) / sqrt(a.sum_u * (a.sum_n + 1) / 12)) AS z_score
FROM R AS r,
     (SELECT sum(sample_u) AS sum_u, sum(sample_n) AS sum_n
      FROM R) AS a
GROUP BY r.sample_u, r.sample_avg, r.sample_n, a.sum_n, a.sum_u
```
In-Database Operations

- Mann-Whitney U Test
  - With large sample count we can get the z_score

```sql
SELECT r.sample_u, r.sample_avg, r.sample_n,
    (r.sample_u - a.sum_u / 2) / \sqrt(a.sum_u * (a.sum_n + 1) / 12) AS z_score
FROM R AS r, (SELECT sum(sample_u) AS sum_u, sum(sample_n) AS sum_n FROM R) AS a
GROUP BY r.sample_u, r.sample_avg, r.sample_n, a.sum_n, a.sum_u
```

This can be easily implemented as a stored procedure:
```sql
SELECT man_whitney(value) FROM table
```
In-Database Operations

- Resampling
  - Sampling several subsets instead of the whole population.
- **Bootstrapping** def= Sampling to buckets (w/ replacement) then combine buckets
- Jackknifing def= Leave some data out and sample; then combine runs
In-Database Operations

- Bootstrapping
- Pre-select which trials contain which rows
- Suppose we have population $N = 100$ and we have subsamples of $M = 3$

```sql
CREATE VIEW design AS
SELECT a.trial_id, floor(100 * random()) AS row_id
FROM generate_series(1, 10000) AS a (trial_id),
generate_series(1, 3) AS b (subsample_id)
```
In-Database Operations

- Bootstrapping
- Calculate the sampling distribution

```
CREATE VIEW trials AS
SELECT d.trial_id, AVG(T.value) AS avg_value
FROM design d, T
WHERE d.row_id = T.row_id
GROUP BY d.trial_id
```
In-Database Operations

- Bootstrapping
- Final distribution parameters

```sql
SELECT AVG(avg_value), STDDEV(avg_value)
FROM trials
```
MAD DBMS

- Loading/Unloading
  - Scatter/Gather Streaming – fast!
- Extract-Transform-Load vs Extract-Load-Transform
- Greenplum has MapReduce support!
MAD DBMS

• Storage

• Tunable table types:
  • external tables (e.g. files)
  • heap tables (frequent updates)
  • append-only tables (rare updates)
  • column-stores flexibility

• All tables have a distribution policy
MAD DBMS

- Partitioning
  - Partition by range of values or columns (list)
    - i.e. partition by timestamp old stuff goes to compressed table, new stuff goes to heap storage.
  - Query optimizer knows the partitioning scheme
  - This is useful for staging data
MAD DBMS

- MAD Programming
  - Use your favorite programming language extension
  - Programmers must think out the code works w/o shared memory. (data-parallel)
  - *Map* and *Reduce* functions can be written in Python, Perl, or R.
  - They run using the Scatter/Gather technique. (Any table type)
  - SQL is not necessary!
MAD DBMS

• Future
  • Need a better repository for the library of functions. (Morpheus SIGMOD06)
  • Automatically choose data and tables layouts automatically
  • Use online aggregation for faster answers to queries.
Questions
Questions

• Why would one be apposed to the M.A.D. paradigm?
Questions

• Why would one be apposed to the M.A.D. paradigm?

• How does MADLib and Greenplum fit into the NoSQL movement?
Questions

• Why would one be apposed to the M.A.D. paradigm?

• How does MADLib and Greenplum fit into the NoSQL movement?

• Is there a market for an AppStore for functions/utilities for MADLib?
Conclusion

• This paper describes the M.A.D. (Magnetic Agile Deep) mindset and the M.A.D. library.

• It demonstrated how to perform analytics inside the DB using SQL and stored procedures.

• It also described the Greenplum database. A general data-parallel database for simple and complex queries from flexible tables.
Thank You!
MADLib Installation (ubuntu)

- sudo apt-get install postgresql flex bison oxygen libboost1.42-* git graphviz libarmadillo-dev libarmadillo-doc
- sudo apt-get install lapack++ blas++
- git clone https://github.com/madlib/madlib.git
- ./configure
- cd build
- cd make

https://github.com/madlib/madlib/wiki/Building-MADlib-from-Source
Matrix Multiplication Demo

-- Create tables in the new matrix format
CREATE TABLE A (row_number INTEGER,
                 column_number INTEGER,
                 value DECIMAL);

INSERT INTO A VALUES
(1, 1, 1),
(1, 2, 2),
(2, 1, 3),
(2, 2, 4);

CREATE TABLE B (row_number INTEGER,
                 column_number INTEGER,
                 value DECIMAL);

INSERT INTO B VALUES
(1, 1, 5),
(1, 2, 6),
(2, 1, 7),
(2, 2, 8);

-- Check to see that the values are multiplying correctly
SELECT A.row_number, B.column_number,
      A.value::text || "\*" || B.value::text
FROM A, B
WHERE A.column_number = B.row_number

-- Add the Groupby to see the sum work
SELECT A.row_number, B.column_number, SUM(A.value * B.value)
FROM A, B
WHERE A.column_number = B.row_number
GROUP BY A.row_number, B.column_number;

DROP TABLE A;
DROP TABLE B;