The MADlib Analytics Library

or MAD Skills, the SQL

Caleb Welton  Florian Schoppmann  Christopher Ré
MADlib
Scalable Machine Learning for Big Data
Traditional analytics pipeline
The MAD approach

Billions of rows in minutes

Time-to-Insights

Data Prep
Model
Score

Enterprise Data
RDBMS
RDBMS
RDBMS
RDBMS
MADlib in *Action*

Hospital Admittance Case Study
MADlib in Action

Step 1:
- Identify high risk patients

Goal:
- High risk patients will be eligible for early admittance and be administered preemptive antibiotics
MADlib in Action

Step 2:
• Build cost model for treatment

Goal:
• Predict expected cost of treatment
• With and without early admittance.
MADlib in Action

Step 3:
• Optimize early admittance based on risk and cost model

Goal:
• Overall hospital costs will be minimized and patients will receive better care.
MADlib cycle of success

Why didn’t I think of that before?
The MADlib Vision

• Academic and industry contributions
• Think of “CRAN for databases”
  – Repository of open-source ML algorithms
  – This time with data parallelism in mind
• Open-Source Framework
Simple Example: Ordinary Least Squares

# SELECT y, x[1] AS x1, x[2] AS x2 FROM data

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10.14</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>11.93</td>
<td>0.69</td>
<td>0.6</td>
</tr>
<tr>
<td>13.57</td>
<td>1.1</td>
<td>0.9</td>
</tr>
<tr>
<td>14.17</td>
<td>1.39</td>
<td>1.2</td>
</tr>
<tr>
<td>15.25</td>
<td>1.61</td>
<td>1.5</td>
</tr>
<tr>
<td>16.15</td>
<td>1.79</td>
<td>1.8</td>
</tr>
</tbody>
</table>

# SELECT (linregr(y, x)).* FROM data

| coef         | {1.7307, 2.2428} |
| r2           | 0.9475           |
| std_err      | {0.3258, 0.0533} |
| t_stats      | {5.3127, 42.0640} |
| p_values     | {6.7681e-07, 4.4409e-16} |
| condition_no | 169.5093         |
Linear Algebra in the Database

\[ \hat{\beta} = (X^T X)^{-1} X^T y \]
Basic Building Block: User-Defined Aggregates

Aggregation phase 1 on each node:
1. Initialize: \((A, b) = (0, 0)\)
2. Transition for all rows:
   \[
   (A, b) = (A, b) + (x \cdot x^T, x \cdot y)
   \]
3. Send \((A, b)\)

map
reduce

Aggregation phase 2 on master node:
1. Merge: \((\bar{A}, \bar{b}) = (\bar{A}, \bar{b}) + (A, b)\)
2. Finalize: \(\hat{\beta} = \text{solve}(\bar{A}, \bar{b}) = \bar{A}^{-1} \cdot \bar{b}\)
Problem solved?

No – not yet.
ML Algorithms Based on SQL?

• Four Representative Challenges
  1. Lack of portable multi-pass iterations
  2. Roots in first-order logic
  3. Lack of language support for linear algebra
  4. Extensible SQL limited to small working sets

Need:
• Abstraction Layers
• A few compromises for user interface
1. Lack of portable multi-pass iterations

- **WITH RECURSIVE** not reliable basis for portability
- User-defined **driver** functions in Python
  - Outer loops not performance-critical
- Compromise:
  Different user interface

```sql
CREATE TEMP TABLE temp

INSERT INTO temp SELECT step(...) FROM ...

SELECT converged(...) FROM temp, ...

SELECT result(...) FROM temp
```
2. Roots in first-order logic

• Queries need be cognizant of database objects
• Emulate higher-order logic by:
  – dynamic execution of templated SQL
  – abstraction-layer support

```java
FunctionHandle dist = args[0].getAs<FunctionHandle>();
return dist(x, y);
```

• Example: Distance or kernel functions
• On PostgreSQL, use of type REGPROC
3. Lack of language support for linear algebra

- C++ Abstraction Layer uses Eigen
- (Dense) Vectors and matrices: DOUBLE PRECISION[]
- Example:

```cpp
AnyType
solve::run(AnyType& args) {
    MappedMatrix A = args[0].getAs<MappedMatrix>();
    MappedColumnVector b = args[1].getAs<MappedColumnVector>();

    MutableMappedColumnVector x = allocateArray<double>(A.cols());
    x = A.colPivHouseholderQr().solve(b);
    return x;
}
```

Performance:
- No unnecessary copying
- No internal type conversion
4. Extensible SQL limited to small working sets

- Tables only portable option for large states
- Access from UDAs slow or impossible
- Example: k-means benefits from explicit point-to-centroid assignments
  - Problematic:
    ```sql
    UPDATE points SET centroid_id = closest(state, coords)
    ```
  - Requires own pass
  - Not allowed in subqueries
  - PostgreSQL legacy
MADlib Architecture

User Interface

“Driver” Functions
(outter loops of iterative algorithms, optimizer invocations)

High-level Abstraction Layer
(iteration controller, convex optimizers, ...)

Row-level Functions
(inner loops of streaming algorithms, convex optimization callbacks, ...)

Low-level Abstraction Layer
(matrix operations, C++ to RDBMS type bridge, ...)

RDBMS Query Processing
(Greenplum, PostgreSQL, ...)

RDBMS Built-in Functions

SQL, generated from specification

Python with templated SQL

Python

C++
Anatomy of an iterative MADlib module

interState = \textbf{Start}(\textit{args})

\textbf{Repeat}

\textit{In parallel for each} segment:

\hspace{1em} intraState = \textbf{Initialize}(\textit{interState})

\hspace{1em} \textbf{For each} row

\hspace{2em} intraState = \textbf{Transit}(\textit{intraState}, \textit{row})

\hspace{1em} \textbf{For each} \textit{intraState}:

\hspace{2em} intraState = \textbf{Merge}(\textit{oldIntraState}, \textit{intraState})

\hspace{1em} interState = \textbf{Finalize}(\textit{intraState})

\textbf{Until} \textbf{Converged}(\textit{interState})

\textbf{Return} \textbf{End}(\textit{interState})
Performance Trends

• Disk I/O is not always the bottleneck
  • Performance tuning is essential
• Overhead for single query very low (fraction of a second)
• Greenplum achieves nearly perfect speedup
Current Modules

Data Modeling

Supervised Learning
• Naive Bayes Classification
• Linear Regression
• Logistic Regression
• Decision Tree
• Random Forest
• Support Vector Machines

Unsupervised Learning
• Association Rules
• k-Means Clustering
• SVD Matrix Factorization
• Parallel Latent Dirichlet Allocation

Descriptive Statistics

Sketch-based Estimators
• CountMin (Cormode-Muthukrishnan)
• FM (Flajolet-Martin)
• MFV (Most Frequent Values)

Support

Array Operations
Conjugate Gradient
Sparse Vectors
Probability Functions
Feature Extraction

Inferential Statistics

Profile
Quantile
Hypothesis tests

Probability Functions
Feature Extraction
Array Operations
Hypothesis tests
Profile
Quantile
Hypothesis tests
Profile
Quantile
Hypothesis tests
My MADlib Experience: A Testimonial.

Christopher Ré, Wisconsin
Towards a Unified Architecture for in-RDBMS Analytics

Xixuan Feng    Arun Kumar    Benjamin Recht    Christopher Ré
Department of Computer Sciences
University of Wisconsin-Madison
{xfeng, arun, brecht, chrisre}@cs.wisc.edu

ABSTRACT
The increasing use of statistical data analysis in enterprise applications has created an arms race among database vendors to offer ever more sophisticated in-database analytics. In the late 1990s and early 2000s, this brought a wave of data mining toolkits into the RDBMS. Several major vendors are again making an effort toward sophisticated in-database analytics with both open source efforts, e.g., the MADlib platform from Greenplum [18], and several projects at major companies.

Reframing Ideas and Code
Conversations with GP (and Oracle) lead us to better position our SIGMOD12 paper.

QA from GP help to transition from paper to deployed code.
MADlib is Open Source

Enhance Wikipedia with extracted facts from the Web (50+TB of data)

hazy.cs.wisc.edu & www.youtube.com/HazyResearch

Learning & Inference run on (GP or Postgres) + MADLib

Critical: it’s free, open, and we can modify it
Testimonial Summary

MADlib

MADlib is open to contributions and open source
Questions?

http://madlib.net

Caleb Welton
Caleb.Welton@emc.com

Florian Schoppmann
Florian.Schoppmann@emc.com

Christopher Ré
chrisre@cs.wisc.edu