Greenplum: MAD Analytics in Practice

MMDS
June 16th, 2010
Warehousing Today
In the Days of Kings and Priests

• Computers and Data: Crown Jewels
• Executives depend on computers
  – But cannot work with them directly
• The DBA “Priesthood”
  – And their Acronymia
    • EDW, BI, OLAP, 3NF
  – Secret functions and techniques, expensive tools
The Architected EDW

Shadow systems

Slow-moving models

Non-standard, in-memory analytics

‘Shallow’ Business Intelligence

Slow-moving data

Departmental warehouses

Static schemas accrete over time

4/26/2010 Confidential
"Welcome to the Petabyte Age"

- TB disks < $100
- Everything is data
- Rise of data-driven culture
  - Very publicly espoused by Google, Netflix, etc.
  - Terraserver, USAspending.gov

The New Practitioners

- Aggressively Datavorous
- Statistically savvy
- Diverse in training, tools
Greenplum Overview
Greenplum Database Architecture

MPP (Massively Parallel Processing) Shared-Nothing Architecture

- **Master Servers**: Query planning & dispatch
- **Segment Servers**: Query processing & data storage
- **Network Interconnect**: Internode communication
- **External Sources**: Loading, streaming, etc.
- **SQL MapReduce**: PL/R, PL/Perl, PL/Python
Key Technical Innovations

- Scatter-Gather Data Streaming
  - Industry leading data loading capabilities

- Online Expansion
  - Dynamically provision new servers with no downtime

- Map-Reduce Support
  - Parallel programming on data for advanced analytics

- Polymorphic Storage
  - Support for both row and column-oriented storage
Benefits of the Greenplum Database Architecture

- **Simplicity**
  - Parallelism is automatic – no manual partitioning required
  - No complex tuning required – just load and query

- **Scalability**
  - Linear scalability to 1000s of cores, on commodity hardware
  - Each node adds storage, query performance, loading performance
  - Example: 6.5 petabytes on 96 nodes, with 17 trillion records

- **Flexibility**
  - Fully parallelism for SQL92, SQL99, SQL2003 OLAP, MapReduce
  - Any schema (star, snowflake, 3NF, hybrid, etc)
  - Rich extensibility and language support (Perl, Python, R, C, etc)
Customer Example: eBay Petabyte-Scale

- **Business Problem**
  - Fraud detection and click-stream analytics

- **Data Size**
  - 6.5 Petabytes of user data
  - Loading 18 TB every day (130 Billion rows)
  - Every click on eBay’s website, 365 days a year
  - 20 Trillion row fact table

- **Hardware**
  - 96-node Sun Data Warehouse Appliance
  - Expansion to go to 192 nodes

- **Benefit**
  - Scalability and price/performance
  - Cost effective complement to Teradata
MAD Analytics
analyze and model in the cloud

get data into the cloud

push results back into the cloud
What will happen?

How can we do better?

What happened where and when?

How and why did it happen?

Facts

Interpretation
Dolan’s Vocabulary of Statistics

• Data Mining focused on individuals
  - Statistical analysis needs more
  - Focus on density methods
• Need to be able to utter statistical sentences
  - And run massively parallel, on Big Data!

1. (Scalar) Arithmetic
2. Vector Arithmetic
   • i.e. Linear Algebra
   • Functions
     • E.g. probability densities
   • Functionals
     • i.e. functions on functions
     • E.g., A/B testing: a functional over densities
• Misc Statistical methods
  • E.g. resampling
• Paper includes parallelizable, stat-like SQL for
  - Linear algebra (vectors/matrices)
  - Ordinary Least Squares (multiple linear regression)
  - Conjugate Gradient (iterative optimization, e.g. for SVM classifiers)
  - Functionals including Mann-Whitney U test, Log-likelihood ratios
  - Resampling techniques, e.g. bootstrapping

• Encapsulated as stored procedures or UDFs
  - Significantly enhance the vocabulary of the DBMS!
• These are examples.
  - Related stuff in NIPS ’06, using MapReduce syntax
• Plenty of research to do here!!
MAD Analytics Examples
Example

What’s the right price for my products?
**What’s the right price for my products?**

<table>
<thead>
<tr>
<th>Date</th>
<th>BasePrice</th>
<th>Display Price</th>
<th>Feature Price</th>
<th>Feature/Display Price</th>
<th>TPR</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-02-24</td>
<td>7.33</td>
<td>6.67</td>
<td>7.33</td>
<td>7.33</td>
<td>7.20</td>
<td>20484.52</td>
</tr>
<tr>
<td>2009-03-10</td>
<td>7.47</td>
<td>5.94</td>
<td>5.72</td>
<td>7.00</td>
<td>5.72</td>
<td>34313.94</td>
</tr>
<tr>
<td>2009-03-24</td>
<td>7.75</td>
<td>6.74</td>
<td>5.74</td>
<td>7.26</td>
<td>5.82</td>
<td>25477.33</td>
</tr>
<tr>
<td>2009-04-07</td>
<td>7.40</td>
<td>7.19</td>
<td>7.40</td>
<td>7.40</td>
<td>7.23</td>
<td>18772.57</td>
</tr>
<tr>
<td>2009-04-21</td>
<td>7.75</td>
<td>7.36</td>
<td>6.74</td>
<td>7.75</td>
<td>6.22</td>
<td>20743.68</td>
</tr>
<tr>
<td>2009-05-05</td>
<td>7.43</td>
<td>6.56</td>
<td>6.98</td>
<td>7.43</td>
<td>5.70</td>
<td>28244.82</td>
</tr>
<tr>
<td>2009-05-19</td>
<td>7.70</td>
<td>6.57</td>
<td>7.70</td>
<td>7.70</td>
<td>6.23</td>
<td>20234.74</td>
</tr>
<tr>
<td>2009-06-02</td>
<td>6.87</td>
<td>6.67</td>
<td>6.87</td>
<td>6.87</td>
<td>6.64</td>
<td>23262.60</td>
</tr>
<tr>
<td>2009-06-16</td>
<td>7.36</td>
<td>7.00</td>
<td>7.36</td>
<td>7.36</td>
<td>7.44</td>
<td>19290.87</td>
</tr>
<tr>
<td>2009-06-30</td>
<td>6.92</td>
<td>6.72</td>
<td>6.92</td>
<td>6.92</td>
<td>6.73</td>
<td>23617.61</td>
</tr>
<tr>
<td>2009-07-14</td>
<td>7.49</td>
<td>7.32</td>
<td>7.49</td>
<td>7.49</td>
<td>7.58</td>
<td>18017.58</td>
</tr>
<tr>
<td>2009-07-28</td>
<td>7.69</td>
<td>7.44</td>
<td>5.69</td>
<td>7.69</td>
<td>5.70</td>
<td>29193.44</td>
</tr>
<tr>
<td>2009-08-25</td>
<td>7.72</td>
<td>6.74</td>
<td>7.72</td>
<td>7.72</td>
<td>5.72</td>
<td>25138.34</td>
</tr>
</tbody>
</table>

**Get the raw data...**

```sql
DROP TABLE IF EXISTS misc.price_promo;
CREATE TABLE misc.price_promo
(
  dt date
  ,base_price numeric
  ,display_price numeric
  ,feature_price numeric
  ,feature_display_price numeric
  ,tpr numeric
  ,volume numeric
) DISTRIBUTED BY(dt);
\copy misc.price_promo from data.csv with delimiter ','
```
What’s the right price for my products?

Train the model...

CREATE TABLE misc.price_promo_coefs AS
SELECT
  coefs[1] AS intercept_beta,
  coefs[2] AS base_price_beta,
  coefs[3] AS display_price_beta,
  coefs[4] AS feature_display_price_beta,
  coefs[5] AS tpr_beta,
r2
FROM (SELECT
  mregr_coef(volume, array[1::int, base_price_per_unit, display_price, feature_display_price, temporary_price_reduction]) AS coefs,
  mregr_r2(volume, array[1::int, base_price_per_unit, display_price, feature_display_price, temporary_price_reduction]) AS r2
FROM misc.price_promo
) AS a
DISTRIBUTED RANDOMLY;

<table>
<thead>
<tr>
<th>intercept_beta</th>
<th>base_price_beta</th>
<th>display_price_beta</th>
<th>feature_display_price_beta</th>
<th>tpr_beta</th>
<th>r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>72804.48332</td>
<td>5049.03841</td>
<td>-1388.842417</td>
<td>-6203.882026</td>
<td>-4801.114351</td>
<td>0.883172235</td>
</tr>
</tbody>
</table>
What’s the right price for my products?

Evaluate the model...

CREATE OR REPLACE VIEW misc.v_price_promo_fitted AS
SELECT
  volume
, volume_fitted
,100 * abs(volume - volume_fitted)::numeric / volume AS ape
FROM ( SELECT
  p.volume
, c.intercept_beta
  + p.base_price * c.base_price_beta
  + p.display_price * c.display_price_beta
  + p.feature_display_price * c.feature_display_price_beta
  + p.tpr * c.tpr_beta
  AS volume_fitted
FROM misc.price_promo_coefsc
  ,misc.price_promop
) AS a
Example

*What are our customers saying about us?*
What are our customers saying about us?

- How do you discern trends and categories within thousands of on-line conversations?
  - Search for relevant blogs
  - Construct a ‘fingerprint’ for each document based on word frequencies
  - Use this to define what it means for documents to be similar, or ‘close’
  - Identify ‘clusters’ of documents
Accessing the data

• Build the directory list into a set of files that we will access:

```
-INPUT:
  NAME: filelist
  FILE:
    - maple:/Users/demo/blogsplog/filelist1
    - maple:/Users/demo/blogsplog/filelist2
  COLUMNS:
    - path text
```

• For each record in the list "open()" the file and read it in its entirety

```
-MAP:
  NAME: read_data
  PARAMETERS: [path text]
  RETURNS: [id int, path text, body text]
  LANGUAGE: python
  FUNCTION:
    (_, fname) = path.rsplit('/', 1)
    (id, _) = fname.split(’.’)
    body = f.open(path).read()
```

<table>
<thead>
<tr>
<th>id</th>
<th>path</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>2482</td>
<td>/Users/demo/blogsplog/model/2482.html</td>
<td>&lt;!DOCTYPE html PUBLIC &quot;...</td>
</tr>
<tr>
<td>1</td>
<td>/Users/demo/blogsplog/model/1.html</td>
<td>&lt;!DOCTYPE html PUBLIC &quot;...</td>
</tr>
<tr>
<td>10</td>
<td>/Users/demo/blogsplog/model/1000.html</td>
<td>&lt;!DOCTYPE html PUBLIC &quot;...</td>
</tr>
<tr>
<td>2484</td>
<td>/Users/demo/blogsplog/model/2484.html</td>
<td>&lt;!DOCTYPE html PUBLIC &quot;...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Parse the documents into word lists

Convert HTML documents into parsed, tokenized, stemmed, term lists with stop-word removal:

-MAP:
   NAME: extract_terms
   PARAMETERS: [id integer, body text]
   RETURNS: [id int, title text, doc _text]
   FUNCTION: |

   if 'parser' not in SD:
      import ...
      class MyHTMLParser(HTMLParser):
         ...
         SD['parser'] = MyHTMLParser()

      parser = SD['parser']
      parser.reset()
      parser.feed(body)

      yield (id, parser.title, '{" + ",".join(parser.doc) + "}')
Parse the documents into word lists

Use the HTMLParser library to parse the html documents and extract titles and body contents:

```python
if 'parser' not in SD:
    from HTMLParser import HTMLParser

    class MyHTMLParser(HTMLParser):
        def __init__(self):
            HTMLParser.__init__(self)

        def handle_data(self, data):
            data = data.strip()
            if self.inhead:
                if self.tag == 'title':
                    self.title = data
            if self.inbody:

        parser = SD['parser']
        parser.reset()

...
Parse the documents into word lists

Use nltk to tokenize, stem, and remove common terms:

```python
if 'parser' not in SD:
    from nltk import WordTokenizer, PorterStemmer, corpus
    ...
    class MyHTMLParser(HTMLParser):
        def __init__(self):
            ...
            self.tokenizer = WordTokenizer()
            self.stemmer = PorterStemmer()
            self.stopwords = dict(map(lambda x: (x, True),
                                       corpus.stopwords.words()))
            ...
        def handle_data(self, data):
            ...
            if self.inbody:
                tokens = self.tokenizer.tokenize(data)
                stems = map(self.stemmer.stem, tokens)
                for x in stems:
                    if len(x) < 4: continue
                    x = x.lower()
                    if x in self.stopwords: continue
                    self.doc.append(x)
                ...
            parser = SD['parser']
            parser.reset()
            ...
```
Parse the documents into word lists

Use nltk to tokenize, stem, and remove common terms:

```python
if 'parser' not in SD:
    from nltk import WordTokenizer, PorterStemmer, corpus
    ...
    class MyHTMLParser(HTMLParser):
        def __init__(self):
            ...
            self.tokenizer = WordTokenizer()
            self.stemmer = PorterStemmer()
            self.stopwords = dict(map(lambda x: (x, True), corpus.stopwords.words()))
        def handle_data(self, data):
            ...
            if self.inbody:
                tokens = self.tokenizer.tokenize(data)
                stems = map(self.stemmer.stem, tokens)
                for x in stems:
                    if len(x) < 4: continue
                    x = x.lower()
                    if x in self.stopwords: continue
                    self.doc.append(x)
        ...
parser = SD['parser']
parser.reset()
...```

```
shell$ gpmapreduce -f blog-terms.yml
mapreduce_75643_run_1
DONE

sql# SELECT id, title, doc FROM blog_terms LIMIT 5;

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>2482</td>
<td>noodlepie</td>
<td>{noodlepi, from, gutter, grub, gourmet, tbl, noodlepi, blog, scoff,...</td>
</tr>
<tr>
<td>1</td>
<td>Bhootakannadi</td>
<td>{bhootakannadi, 2005, unifi, feed, gener, comment, final, integr,...</td>
</tr>
<tr>
<td>10</td>
<td>Tea Set</td>
<td>{novelti, dish, goldilock, bear, bowl, lide, contain, august,...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
...```
Create histograms of word frequencies

Extract a term-dictionary of terms that show up in at least ten blogs

```
sql#SELECT term, sum(c) AS freq, count(*) AS num_blogs
    FROM ( 
        SELECT id, term, count(*) AS c 
        FROM ( 
            SELECT id, unnest(doc) AS term 
            FROM blog_terms 
        ) term_unnest 
        GROUP BY id, term 
    ) doc_terms 
    WHERE term IS NOT NULL 
GROUP BY term 
HAVING count(*) > 10;
```

<table>
<thead>
<tr>
<th>term</th>
<th>freq</th>
<th>num_blogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>sturdi</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>canon</td>
<td>97</td>
<td>40</td>
</tr>
<tr>
<td>group</td>
<td>48</td>
<td>17</td>
</tr>
<tr>
<td>skin</td>
<td>510</td>
<td>152</td>
</tr>
<tr>
<td>linger</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>blunt</td>
<td>20</td>
<td>17</td>
</tr>
</tbody>
</table>
Create histograms of word frequencies

Use the term frequencies to construct the term dictionary…

```
sql# SELECT array(SELECT term FROM blog_term_freq) dictionary;
```

dictionary

```
(sturdi,canon,group,skin,linger,blunt,detect,giver,annoy,telephon,...)
```

…then use the term dictionary to construct feature vectors for every document, mapping document terms to the features in the dictionary:

```
sql# SELECT id, gp_extract_feature_histogram(dictionary, doc)
               FROM blog_terms, blog_features;
```

<table>
<thead>
<tr>
<th>id</th>
<th>term_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2482</td>
<td>{3,1,37,1,18,1,29,1,45,1,...}:{0,2,0,4,0,1,0,1,0,1,...}</td>
</tr>
<tr>
<td>1</td>
<td>{41,1,34,1,22,1,125,1,387,...}:{0,9,0,1,0,1,0,3,0,2,...}</td>
</tr>
<tr>
<td>10</td>
<td>{3,1,4,1,30,1,18,1,13,1,4,...}:{0,2,0,6,0,12,0,3,0,1,0,1,...}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Create histograms of word frequencies

Format of a sparse vector

<table>
<thead>
<tr>
<th>id</th>
<th>term_count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>{3,1,40,...} {0,2,0,...}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Dense representation of the vector

<table>
<thead>
<tr>
<th>id</th>
<th>term_count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>{0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

dictionary

{sturdi,canon,group,skin linger,blunt,detect,giver,...}

Representing the document

{skin, skin, ...}
Transform the blog terms into statistically useful measures

Use the feature vectors to construct **TFxIDF** vectors:

These are a measure of the importance of terms.

```sql
# SELECT id, (term_count*logidf) tfidf
FROM blog_histogram, (  
   SELECT log(count(*)/count_vec(term_count)) logidf
   FROM blog_histogram
) blog_logidf;

<table>
<thead>
<tr>
<th>id</th>
<th>tfidf</th>
</tr>
</thead>
<tbody>
<tr>
<td>2482</td>
<td>{3,1,37,1,29,1,45,1,...}:{0,8.25206814635817,0,0.34311110...}</td>
</tr>
<tr>
<td>1</td>
<td>{41,1,34,1,22,1,125,1,387,...}:{0,0.771999985977529,0,1.999427...}</td>
</tr>
<tr>
<td>10</td>
<td>{3,1,4,1,30,1,18,1,13,1,4,...}:{0,2.95439664949608,0,3.2006935...}</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
```
Create document clusters around iteratively defined centroids

Now that we have TFxIDFs we have something that is a statistically significant metric, which enables all sorts of real analytics.

The current example is k-means clustering which requires two operations.

First, we compute a distance metric between the documents and a random selection of centroids, for instance cosine similarity:

```sql
SELECT id, tfxidf, cid, 
    ACOS((tfxidf %*% centroid) / 
        (svec_l2norm(tfxidf) * svec_l2norm(centroid))) 
) AS distance
FROM blog_tfxidf, blog_centroids;
```

<table>
<thead>
<tr>
<th>id</th>
<th>tfxidf</th>
<th>cid</th>
<th>distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2482</td>
<td>{3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.3431111...}</td>
<td>1</td>
<td>1.53672977</td>
</tr>
<tr>
<td>2482</td>
<td>{3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.3431111...}</td>
<td>2</td>
<td>1.55720354</td>
</tr>
<tr>
<td>2482</td>
<td>{3,1,37,1,18,1,29,1,45,1,...}:{0,8.25206814635817,0,0.3431111...}</td>
<td>3</td>
<td>1.55040145</td>
</tr>
</tbody>
</table>
Create document clusters around iteratively defined centroids

Next, use an averaging metric to re-center the mean of a cluster:

sql# SELECT cid, sum(tfidf)/count(*) AS centroid
FROM (SELECT id, tfidf, cid, row_number() OVER (PARTITION BY id ORDER BY distance, cid) rank
FROM blog_distance
) blog_rank
WHERE rank = 1
GROUP BY cid;

<table>
<thead>
<tr>
<th>cid</th>
<th>centroid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>{1,1,1,1,1,1,1,1,1,...}:{0.157556041103536,0.0635233900749665,0.050...}</td>
</tr>
<tr>
<td>2</td>
<td>{1,1,1,1,1,1,3,1,1,...}:{0.0671131209568817,0.332220028552986,0,0...}</td>
</tr>
<tr>
<td>1</td>
<td>{1,1,1,1,1,1,1,1,1,...}:{0.103874521481016,0.158213686890834,0.0540...}</td>
</tr>
</tbody>
</table>

Repeat the previous two operations until the centroids converge, and you have k-means clustering.
MAD Analytics in Practice
MAD Skills in practice

- Extracted data from EDW and other source systems into new analytic sandbox
- Generated a social graph from call detail records and subscriber data
- Within 2 weeks uncovered behavior where “connected” subscribers were seven times more likely to churn than average user
Retention models

- Customer Retention
  - Identify those at risk of abandoning their accounts.
  - Use logistic regression models, or SAS scoring models.

- Also used to predict...
  - fraud in on-line and financial transactions
  - hospital return visits
  - etc.

### Credit Card Number vs Probability of Fraud

<table>
<thead>
<tr>
<th>Credit Card Number</th>
<th>Probability of Fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>7125 6289 5972 9510</td>
<td>15%</td>
</tr>
<tr>
<td>3955 8125 1327 7120</td>
<td>22%</td>
</tr>
<tr>
<td>2190 6379 9218 9290</td>
<td>7%</td>
</tr>
<tr>
<td>2760 1924 2864 0950</td>
<td>47%</td>
</tr>
<tr>
<td>4915 1908 8302 9940</td>
<td>12%</td>
</tr>
<tr>
<td>3534 7203 6200 4010</td>
<td>3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Probability of Churn</th>
</tr>
</thead>
<tbody>
<tr>
<td>611-43-2435</td>
<td>15%</td>
</tr>
<tr>
<td>812-35-1035</td>
<td>22%</td>
</tr>
<tr>
<td>253-23-2943</td>
<td>7%</td>
</tr>
<tr>
<td>732-62-1435</td>
<td>47%</td>
</tr>
<tr>
<td>483-32-5298</td>
<td>12%</td>
</tr>
<tr>
<td>821-90-8574</td>
<td>3%</td>
</tr>
</tbody>
</table>
Segmentation

Using segments

– Create clusters of customers based on profiles, product usage, etc.
Association Rules

Using segments

– For low or medium-value customers, compute possible new products using association rules

Product A
Product B
Product X
Product Y
Product Z
Segmentation and Association Rules

Using segments

- Filter down to products associated with high-value customers in the same segment.

Product A
Product B

Product X
Product Y
Product Z
Questions