Dot-Product Join: Scalable In-Database Linear Algebra for Big Model Analytics

Chengjie Qin\textsuperscript{1} and Florin Rusu\textsuperscript{2}

\textsuperscript{1}GraphSQL, Inc.
\textsuperscript{2}University of California Merced

June 29, 2017
Machine Learning (ML) Is Booming

- General frameworks with ML libraries: Hadoop’s Mahout, Spark’s MLLib, GraphLab
- Specialized ML systems: Vowpal Wabbit, SystemML, SimSQL, TensorFlow
- In-Database ML libraries: MADlib, Bismarck, GLADE
Agenda

- Big Model Analytics
- Gradient Descent Optimization
- Big Model Dot-Product
- Dot-Product Join Operator
  - Vector Reordering
  - Batch Execution
  - Gradient Descent Integration
- Conclusions
Recommender Systems

Spotify applies low-rank matrix factorization (LMF) to 24 million users and 20 million songs which is **4.4 billion features** at a relatively small rank of 100.

---

http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify
Text Analytics

<table>
<thead>
<tr>
<th>Full sentence</th>
<th>It does not, however, control whether an exaction is within Congress’s power to tax.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigrams</td>
<td>“It”; “does”; “not,”; “however,”; “control”; “whether”; “an”; “exaction”; “is”; “within”; “Congress’s”; “power”; “to”; “tax.”</td>
</tr>
<tr>
<td>Bigrams</td>
<td>“It does”; “does not,”; “not, however,”; “however, control”; “control whether”; “whether an”; “an exaction”; “exaction is”; “is within”; “within Congress’s”; “Congress’s power”; “power to”; “to tax.”</td>
</tr>
<tr>
<td>Trigrams</td>
<td>“It does not”; “does not, however”; “not, however, control”; “however, control whether”; “control whether an”; “whether an exaction”; “an exaction is”; “exaction is within”; “is within Congress’s”; “within Congress’s power”; “Congress’s power to”; “power to tax.”</td>
</tr>
</tbody>
</table>

Big Model Motivation

In the Cloud …

- There is always enough memory
- You can always add more servers

ML in IoT, Edge, and Fog Environments

- Push processing to the devices acquiring the data which have rather scarce resources
- Data transfer is not a viable alternative for bandwidth and privacy reasons
- Secondary storage (disk, SSD, or flash) is plentiful
Support for Big Models

Existing ML Systems Do Not Support Big Models

- Model is an in-memory container data structure, e.g., vector or map
- Model is array attribute in a single-column table (at most 1 GB in PostgreSQL) and in-memory state of a UDA (User-Defined Aggregate)

Parameter Server [M. Li et al., “Scaling Distributed Machine Learning with the Parameter Server”, OSDI 2014]

- Partition the model across the distributed shared memory of multiple servers, with each server storing a sufficiently small model partition that fits in its local memory
- Extensive hardware investment and considerable network traffic
- Complex partitioning, replication, and synchronization
- Targeted to the Cloud
In-Database Big Model ML

Problem

• Provide efficient and cost-effective in-database support for Big Model ML

Challenge

• ML is heavily based on in-memory linear algebra (ordered arrays)
• Databases are based on disk relational algebra (unordered relations)

Solution

• Offload Big Model to secondary storage and leverage database processing techniques
• Design specialized in-database linear algebra operators for Big Model ML
Agenda

• Big Model Analytics

• **Gradient Descent Optimization**

• Big Model Dot-Product

• Dot-Product Join Operator
  – Vector Reordering
  – Batch Execution
  – Gradient Descent Integration

• Conclusions
ML for Generalized Linear Models

Sparse Matrix Vector Multiplication (SpMV)

- Model is $d$-dimensional vector $\vec{w}$, $d \geq 1$
- Training data $\vec{X}$ of $N$ $d$-dimensional feature vectors $\vec{x}_i$ and their corresponding label $y_i$, $1 \leq i \leq N$
- Objective function (or loss): $\Lambda(\vec{w}) = \min_{\vec{w} \in \mathbb{R}^d} \sum_{i=1}^{N} f(\vec{w}, \vec{x}_i; y_i)$
- Find model $\vec{w}$ that minimizes objective function based on training data

Logistic Regression (LR)

- $\Lambda_{LR}(\vec{w}) = \sum_{i=1}^{N} \log \left( 1 + e^{-y_i \vec{w} \cdot \vec{x}_i} \right)$

Low-Rank Matrix Factorization (LMF)

- $\Lambda_{LMF}(L, R) = \frac{1}{2} \sum_{(i, j) \in M} \left( L_i^T \cdot R_j - M_{ij} \right)^2$
Gradient Descent Optimization

\[
\min_{\vec{w} \in \mathbb{R}^d} \left\{ \Lambda(\vec{w}) = \sum_{i=1}^{N} f(\vec{w}, \vec{x}_i; y_i) \right\}
\]

\[
\vec{w}^{(k+1)} = \vec{w}^{(k)} - \alpha^{(k)} \nabla \Lambda(\vec{w}^{(k)})
\]

\(\alpha^{(k)}\) is step size or learning rate

\(\vec{w}^{(0)}\) is the starting point (random)

\[
\nabla \Lambda(\vec{w}) = \left[ \frac{\partial \Lambda(\vec{w})}{\partial w_1}, \ldots, \frac{\partial \Lambda(\vec{w})}{\partial w_d} \right]
\]

is the gradient

\[
\frac{\partial \Lambda_{LR}(\vec{w})}{\partial w_i} = \sum_{i=1}^{N} \left( -y_i \frac{e^{-y_i \vec{w} \cdot \vec{x}_i}}{1 + e^{-y_i \vec{w} \cdot \vec{x}_i}} \right) \vec{x}_i
\]

\[
\frac{\partial \Lambda_{LMF}(L, R)}{\partial \vec{L}_{i'}} = \sum_{(i', j) \in M} \left( \vec{L}_{i'}^T \cdot \vec{R}_j - M_{i'j} \right) \vec{R}_j^T
\]

- Convergence to minimum guaranteed for convex objective function
Batch Gradient Descent (BGD)

Input: $\{((\vec{x}_j, y_j))_{1 \leq j \leq N}, f, \Lambda, \nabla \Lambda, \vec{w}^{(0)}, \alpha^{(0)}\}$

Output: $\vec{w}^{(k-1)}$

1. Let $k = 1$
2. while (true) do
3. if convergence($\{\Lambda(\vec{w}^{(l)})\}_{0 \leq l < k}$) then break
4. Compute gradient: $\nabla \Lambda(\vec{w}^{(k-1)})\{((\vec{x}_j, y_j))_{1 \leq j \leq N}$
5. Determine step size $\alpha^{(k)}$
6. Update model: $\vec{w}^{(k)} = \vec{w}^{(k-1)} - \alpha^{(k)} \nabla \Lambda(\vec{w}^{(k-1)})$
7. Let $k = k + 1$
8. end while
9. return $\vec{w}^{(k-1)}$

Gradient $\nabla \Lambda$ is standard SpMV between training data $\vec{X}$ and model $\vec{w}$, i.e., $\vec{X} \cdot \vec{w}$
Stochastic Gradient Descent (SGD)

Input: $\{(\vec{x}_j, y_j)\}_{1 \leq j \leq N}$, $f$, $\nabla f$, $\vec{w}^{(0)}$, $\alpha^{(0)}$

Output: $\vec{w}^{(k-1)}$

1. Let $k = 1$
2. while (true) do
3.  if convergence($\{\Lambda(\vec{w}^{(l)})\}_{0 \leq l < k}$) then break
4.  for each example $(\vec{x}_{\eta(i)}, y_{\eta(i)})$ do
5.   Approximate gradient: $\nabla f \left( \vec{w}^{(k)}_{(i-1)}, \vec{x}_{\eta(i)}; y_{\eta(i)} \right)$
6.   $\vec{w}^{(k)}_{(i)} = \vec{w}^{(k)}_{(i-1)} - \alpha^{(k)} \nabla f \left( \vec{w}^{(k)}_{(i-1)}, \vec{x}_{\eta(i)}; y_{\eta(i)} \right)$
7.  end for
8. Update step size $\alpha^{(k)}$
9. Let $k = k + 1$
10. end while
11. return $\vec{w}^{(k-1)}$

Approximate gradient $\nabla f$ is standard vector dot-product between each training vector $\vec{x}_i$ and model $\vec{w}$, i.e., $\vec{x}_i \cdot \vec{w}$
**BGD vs. SGD**

**BGD**

\[ \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha^{(k)} \nabla \Lambda \left( \mathbf{w}^{(k)} \right) \]

- Exact gradient computation
- Single step for one iteration
- Faster convergence close to minimum

**SGD**

\[ \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \beta^{(k)} \nabla f \left( \mathbf{w}^{(k)}, \mathbf{x}^{(k)}; y^{(k)} \right) \]

- Approximate gradient at data point
- One step for each random data point
- Faster convergence far from minimum

Chengjie Qin and Florin Rusu – The Dot-Product Join Operator
Agenda

- Big Model Analytics
- Gradient Descent Optimization
- **Big Model Dot-Product**
  - Dot-Product Join Operator
    - Vector Reordering
    - Batch Execution
    - Gradient Descent Integration
- Conclusions
The Big Model Dot-Product Problem

• Set of (very) sparse vectors, i.e., matrix, $U = \{\vec{u}_1, \ldots, \vec{u}_N\}$

• Dense vector $V = \vec{v}$ that cannot fit in the available memory

• Compute $DP = U \cdot V$ in non-blocking mode, i.e., generate $DP_i = \vec{u}_i \cdot \vec{v}$ one-by-one
  
  – Required for SGD where $V$ is updated after each $DP_i$ is computed

• Novelty: Constrained SpMV with $V$ stored on disk
Relational Dot-Product

- Store sparse matrix $U$ and dense vector $V$ as two relational tables
  - $U$(index INTEGER, value NUMERIC, tid INTEGER)
    - $\{(1,1,1); (3,3,1); (4,9,1); \ldots\}$
  - $V$(index INTEGER, value NUMERIC)
    - $\{(1,0.1); (2,0.5); (3,0.3); (4,0.2); (5,0.1); (6,0.1)\}$
- SQL query to compute $DP$ is blocking
  - $\text{SELECT U.tid, SUM(U.value*V.value) FROM U, V WHERE U.index=V.index GROUP BY U.tid}$
- Vectors are represented as tuples which incurs redundancy

**Relational dot-product**

$U \times V \rightarrow DP$

$U\text{.index} = V\text{.index}$

$\text{U.tid; SUM(U.value*V.value)}$

Chengjie Qin and Florin Rusu – The Dot-Product Join Operator
Array-Relation Join

- Store sparse matrix $U$ in the coordinate representation with two ARRAY attributes for the non-zero index and the corresponding value
  
- $U(\text{index INTEGER}[,], \text{value NUMERIC}[,], \text{tid INTEGER})$
  
  - $\{(1,3,4), [1,3,9], 1); \ldots\}^{...}$

- $V(\text{index INTEGER}, \text{value NUMERIC})$
  
  - $\{(1,0.1); (2,0.5); (3,0.3); (4,0.2); (5,0.1); (6,0.1)\}$

- SQL query to compute $DP$ is blocking

SELECT $U.tid,$
SUM($U.value[idx(U.index,V.index)]*V.value)$
FROM $U,$ $V$
WHERE $V.index = \text{ANY}(U.index)$
GROUP BY $U.tid$

- Intermediate join size can be very large

Chengjie Qin and Florin Rusu – The Dot-Product Join Operator
Agenda

• Big Model Analytics
• Gradient Descent Optimization
• Big Model Dot-Product
• **Dot-Product Join Operator**
  – Vector Reordering
  – Batch Execution
  – Gradient Descent Integration
• Conclusions
Dot-Product Join Operator – Idea

Array-Relation Join

Dot-Product Join

• Push group-by aggregate into join and generate $DP_i$ cells one-at-a-time by streaming $U$ page-at-a-time and probing $V$ (non-blocking)

• Each $DP_i$ cell can be computed in memory

• $V$ is range-based partitioned
Dot-Product Join Operator – Approach

- Minimize number of secondary storage page accesses to vector $V$
  - Reorder vectors $\vec{u}_i \in U$ at page-level
- Minimize number of requests for cells in $V$
  - Batch reordered vectors $\vec{u}_i \in U$ and emit a single request for all the accessed pages
  - Enhance buffer manager with set requests
Dot-Product Join Operator – Algorithm

Require:

U (index INTEGER[], value NUMERIC[], tid INTEGER)
V (index INTEGER, value NUMERIC)
memory budget \( M \)

Ensure: \( \text{DP (tid INTEGER, product NUMERIC)} \)

1. for each page \( u_p \in U \) do

OPTIMIZATION

2. Reorder vectors \( \vec{u}_i \) to cluster similar vectors together
3. Group vectors \( \vec{u}_i \) into batches \( B_j \) that access at most \( M \) pages from \( V \)

BATCH EXECUTION

4. for each batch \( B_j \) do

5. Collect pages \( v_p \in V \) accessed by vectors \( \vec{u}_i \in B_j \) into a set \( v_{B_j} = \{ v_p \in V | \exists \vec{u}_i \in B_j \text{ that accesses } v_p \} \)
6. Request access to pages in \( v_{B_j} \)

DOT-PRODUCT COMPUTATION

7. \( dp_i \leftarrow \text{Dot-Product}(\vec{u}_i, V) \)
8. return \( dp_i \)

end for

end for
Vector Reordering

- Minimize number of secondary storage page accesses to vector $V$
- Find optimal order is NP-hard (minimum Hamiltonian path)
- Reverse Cuthill-McKee algorithm (RCM) in SpMV permutes both rows and columns
- Cluster similar vectors together
- 3 heuristics to compute a good ordering for in-memory working set (inspector-executor paradigm in SpMV)
  - K-Center Clustering
  - Locality-Sensitive Hashing (LSH)-based Nearest Neighbor
  - Radix Sort

![Diagram showing page accesses and requests before and after reordering.](image)
**K-Center Clustering**

**Require:** Vectors \( \{\vec{u}_1, \ldots, \vec{u}_N\} \) with page requests

**Ensure:** Reordered input vectors \( \{\vec{u}_{i1}, \ldots, \vec{u}_{iN}\} \)

1. Initialize first set of \( k \) centers with random vectors \( \vec{u}_i \)
2. Assign each vector to the center having the minimum set difference cardinality
3. Let \( X_l \) be the set of vectors assigned to center \( l, 1 \leq l \leq k \)
4. Call *K-Center Reordering* recursively for the sets \( X_l \) with requests that do not fit in memory
5. Reorder centers and their corresponding vectors

- Complexity: \( O(kdN) \); \( k \) centers, \( N \) \( d \)-dimensional vectors
- Only partial ordering between two vectors
- Randomized algorithm sensitive to initialization
LSH-based Nearest Neighbor

Require:
Vectors \( \{ \vec{u}_1, \ldots, \vec{u}_N \} \) with page requests
\( m \) minwise hash functions grouped into \( b \) bands

Ensure: Reordered input vectors \( \{ \vec{u}_{i1}, \ldots, \vec{u}_{iN} \} \)

Compute LSH tables

LSH-based nearest neighbor search
1. Initialize \( \vec{u}_{i1} \) with a random vector \( \vec{u}_i \)
2. for \( j = 1 \) to \( N - 1 \) do
3. Let \( X_k \) be the set of vectors co-located in the same bucket with \( \vec{u}_{ij} \) in hash table \( Hash_k \) and not selected
4. \( X \leftarrow X_1 \cup \cdots \cup X_b \)
5. Let \( \vec{u}_{ij+1} \) be the vector in \( X \) with the minimum set difference cardinality \( |C_{ij+1,ij}| \) to the current vector \( \vec{u}_{ij} \)
6. end for

- Complexity: \( O(mdN) \); \( m \) hash functions, \( N \) \( d \)-dimensional vectors
- Only partial ordering between two vectors
- Randomized algorithm sensitive to initialization
Radix Sort

Require: Vectors \( \{ \vec{u}_1, \ldots, \vec{u}_N \} \) with page requests
Ensure: Reordered input vectors \( \{ \vec{u}_i_1, \ldots, \vec{u}_i_N \} \)

Page request frequency computation
1. Compute page request frequency across vectors \( \{ \vec{u}_1, \ldots, \vec{u}_N \} \)
2. for each vector \( \vec{u}_i \) do
3. Represent \( \vec{u}_i \) by a bitset of 0’s and 1’s where a 1 at index \( k \) corresponds to the vector requesting page \( k \)
4. Reorder the bitset in decreasing order of the page request frequency, i.e., index 1 corresponds to the most frequent page
5. end for

Radix sort
6. Apply radix sort to the set of bitsets
7. Let \( \vec{u}_{i_j} \) be the vector corresponding to the bitset at position \( j \) in the sorted order

- Complexity: \( \mathcal{O}(dN) \); \( N \) \( d \)-dimensional vectors
- Strict ordering between two vectors based on a single dimension at a time
Heuristics Experimental Comparison

- Setup: 16 cores, 28 GB of memory, and 1 TB 7200 RPM SAS hard-drive (120 MB/s)

![Graphs showing reordering times and relative improvements](image)

- Average reordering time per page
- Relative improvement over LRU

- Reordering time
  - Radix sort is the only scalable solution (less than 1 second for $2^{16}$ vectors)

- LRU comparison
  - LSH provides largest reduction over LRU at almost 30% for large pages
Batch Execution

- Minimize number of requests for cells in $V$
- Construct batches by iterating over reordered vectors until memory budget for pages in $V$ is full (optimal)
- Page accesses from same batch are grouped into a single request
- Enhanced buffer manager with page set requests

# Page Accesses: 4, # Requests: 16

# Page Accesses: 4, # Requests: 6
Batch Execution Evaluation

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Dims</th>
<th># Examples</th>
<th>Size</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>1B</td>
<td>80K</td>
<td>4.2 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>skewed</td>
<td>1B</td>
<td>1M</td>
<td>4.5 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>splice</td>
<td>13M</td>
<td>500K</td>
<td>30 GB</td>
<td>100 MB</td>
</tr>
</tbody>
</table>

- Radix sort reordering, page size is 4096
- Memory budget is 20% of model size (vector $V$ dimensionality)

Average dot-product join time per page

Dot-product join time over skewed

Chengjie Qin and Florin Rusu – The Dot-Product Join Operator
Gradient Descent Integration

- SGD includes gradient computation and model update beyond dot-product

![Graphs showing execution time for different available memory scenarios for 'splice' and 'matrix' datasets with Dot-product and SGD methods.]
# Dot-Product Join vs. Alternatives

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Dims</th>
<th># Examples</th>
<th>Size</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>1B</td>
<td>80K</td>
<td>4.2 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>skewed</td>
<td>1B</td>
<td>1M</td>
<td>4.5 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>matrix</td>
<td>10M x 10K</td>
<td>300M</td>
<td>4.5 GB</td>
<td>80 GB</td>
</tr>
<tr>
<td>splice</td>
<td>13M</td>
<td>500K</td>
<td>30 GB</td>
<td>100 MB</td>
</tr>
<tr>
<td>MovieLens</td>
<td>6K x 4K</td>
<td>1M</td>
<td>24 MB</td>
<td>80 MB</td>
</tr>
</tbody>
</table>

- Memory budget is 40% of model size (vector $V$ dimensionality) for GLADE and Dot-Product Join; 10 GB in PostgreSQL (PG); unlimited in MonetDB and SciDB

- Execution time (in seconds) for relational (R) and array (A) solutions; N/A stands for not finishing in 24 hours

<table>
<thead>
<tr>
<th></th>
<th>GLADE R</th>
<th>PG R</th>
<th>PG A</th>
<th>MonetDB R</th>
<th>SciDB A</th>
<th>DOT-PRODUCT JOIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>37851</td>
<td>N/A</td>
<td>N/A</td>
<td>4280</td>
<td>N/A</td>
<td>3914</td>
</tr>
<tr>
<td>skewed</td>
<td>13000</td>
<td>N/A</td>
<td>N/A</td>
<td>4952</td>
<td>N/A</td>
<td>786</td>
</tr>
<tr>
<td>matrix</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>9045</td>
</tr>
<tr>
<td>splice</td>
<td>9554</td>
<td>81054</td>
<td>N/A</td>
<td>2947</td>
<td>N/A</td>
<td>6769</td>
</tr>
<tr>
<td>MovieLens</td>
<td>905</td>
<td>5648</td>
<td>72480</td>
<td>N/A</td>
<td>1116</td>
<td>477</td>
</tr>
</tbody>
</table>
Conclusions

• Investigate the Big Model analytics problem and identify dot-product as the critical operation in training generalized linear models

• Establish a direct correspondence with the sparse matrix vector (SpMV) multiplication problem

• Present several alternatives for implementing dot-product in a relational database

• Design the first array-relation dot-product join database operator targeted at secondary storage and introduce two optimizations – dynamic batch processing and reordering – to make the operator practical

• Devise three batch reordering heuristics – K-center, LSH, and Radix – inspired from optimizations to the SpMV kernel and evaluate them thoroughly.

• Execute an extensive set of experiments that evaluate each sub-component of the operator and compare our overall solution with alternative dot-product implementations over synthetic and real data

• Dot-product join provides significant reduction in execution time over alternative in-database solutions
Thank you.

Questions ????