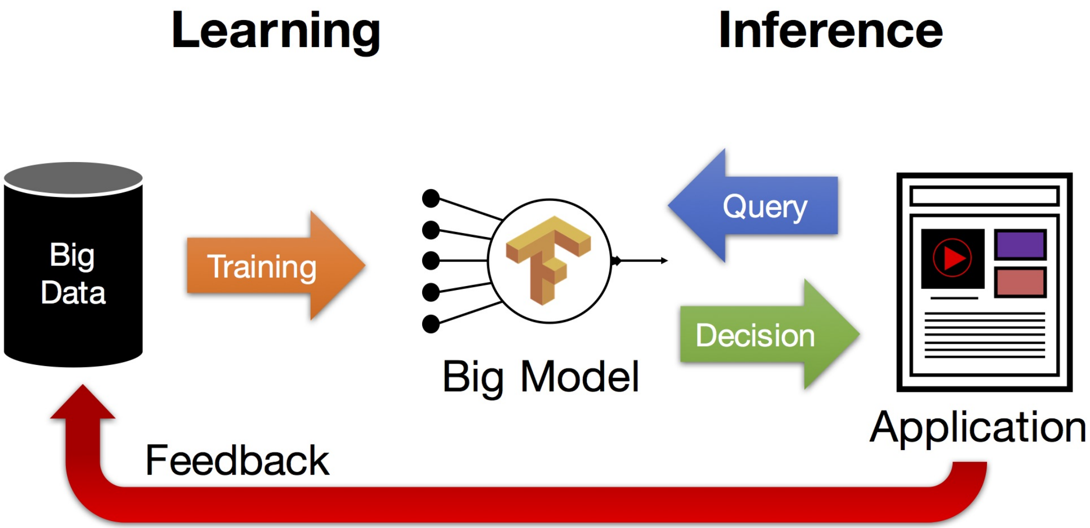
Dot-Product Join: Scalable In-Database Linear Algebra for Big Model Analytics

Chengjie Qin[1] and **Florin Rusu**[2]

[1]GraphSQL, Inc.
[2]University of California Merced

June 29, 2017

Machine Learning (ML) Is Booming



https://ucbrise.github.io/cs294-rise-fa16/prediction_serving.html

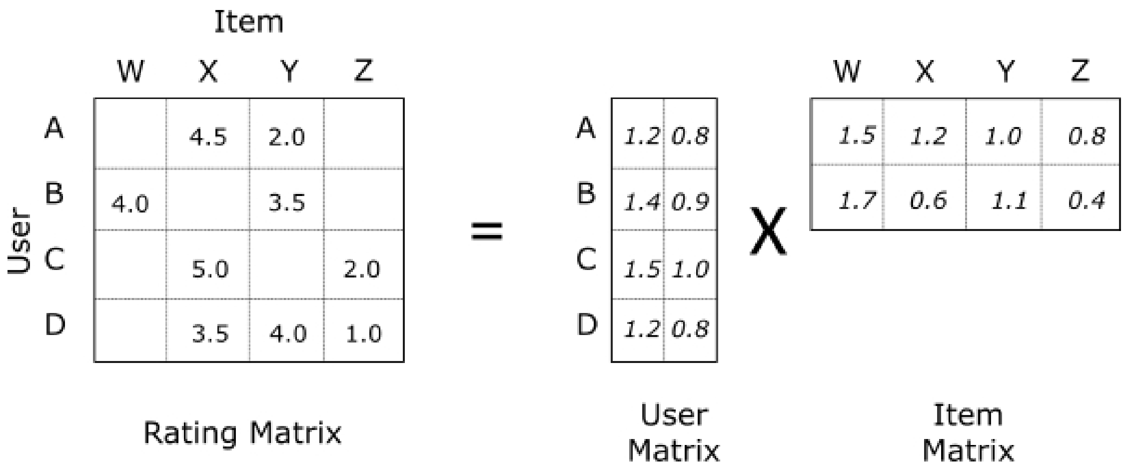
- General frameworks with ML libraries: Hadoop's Mahout, Spark's MLLib, GraphLab
- Specialized ML systems: Vowpal Wabbit, SystemML, SimSQL, TensorFlow
- **In-Database ML libraries: MADlib, Bismarck, GLADE**

Agenda

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

Big Model Example 1

Recommender Systems



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

- 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

Big Model Example 2

Text Analytics

Full sentence	It does not, however, control whether an exaction is within Congress's power to tax.
Unigrams	"It"; "does"; "not,"; "however,"; "control"; "whether"; "an"; "exaction"; "is"; "within"; "Congress's"; "power"; "to"; "tax."
Bigrams	"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."
Trigrams	"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."

N-gram features of a sentence

- For the English Wikipedia corpus, a feature vector with **25 billion unigrams and 218 billion bigrams** can be constructed [S. Lee, J. K. Kim et al., "On Model Parallelization and Scheduling Strategies for Distributed Machine Learning", NIPS 2015]

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

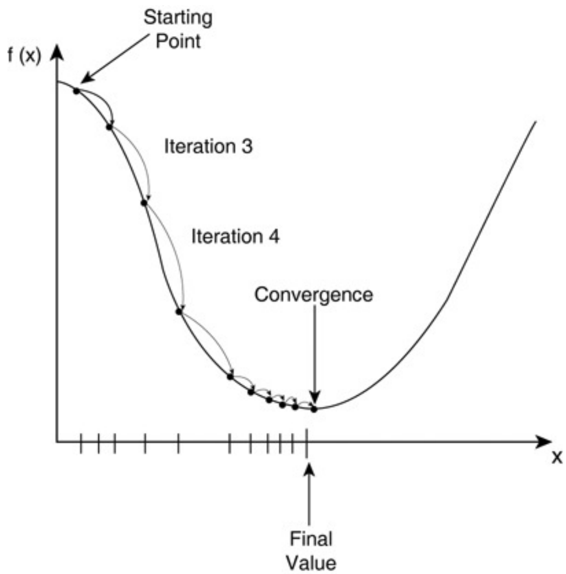
Logistic Regression (LR)

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

Low-Rank Matrix Factorization (LMF)

- $\Lambda_{LMF}(L,R) = \frac{1}{2} \sum_{(i,j) \in M} \left(\vec{L}_i^T \cdot \vec{R}_j - M_{ij}\right)^2$

Gradient Descent Optimization



http://www.yaldex.com/game-development/1592730043_ch18lev1sec4.html

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

$$\vec{w}^{(k+1)}=\vec{w}^{(k)}-\alpha^{(k)}\nabla\Lambda\left(\vec{w}^{(k)}\right)$$

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

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

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

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

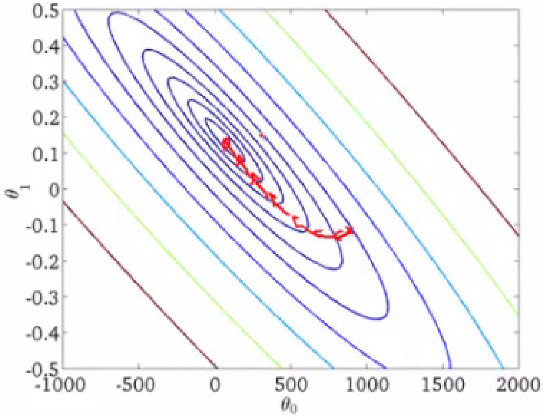
$$\frac{\partial\Lambda_{LMF}(L,R)}{\partial\vec{L}_{i'}}=\sum_{(i',j)\in M}\left(\vec{L}_{i'}^T\cdot\vec{R}_j-M_{i'j}\right)\vec{R}_j^T$$

- Convergence to minimum guaranteed for convex objective function

Batch Gradient Descent (BGD)

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

Let $k = 1$
while (**true**) **do**
  if convergence($\{\Lambda(\vec{w}^{(l)})\}_{0 \le l < k}$) **then break**
  Compute gradient: $\nabla\Lambda(\vec{w}^{(k-1)})\{(\vec{x}_j, y_j)\}_{1 \le j \le N}$
  Determine step size $\alpha^{(k)}$
  Update model: $\vec{w}^{(k)} = \vec{w}^{(k-1)} - \alpha^{(k)} \nabla\Lambda(\vec{w}^{(k-1)})$
  Let $k = k + 1$
end while
return $\vec{w}^{(k-1)}$



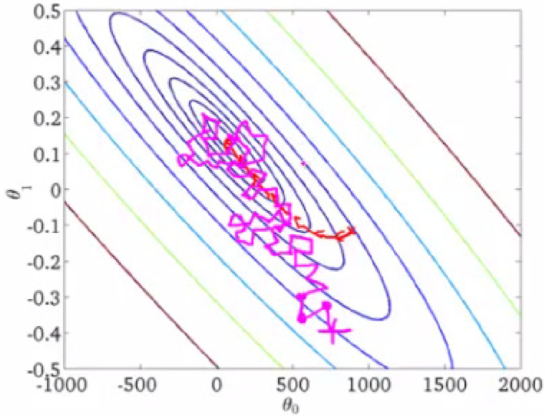
http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

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

Stochastic Gradient Descent (SGD)

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

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

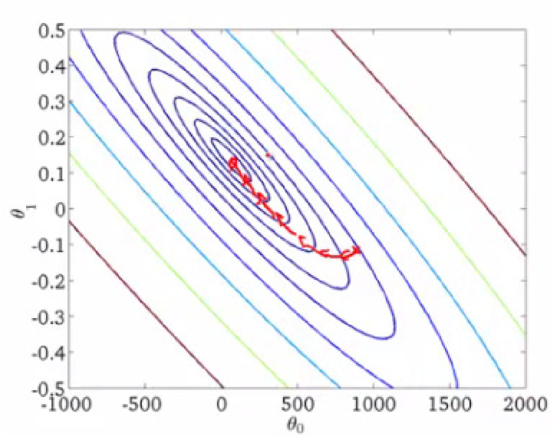


http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

Approximate gradient ∇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

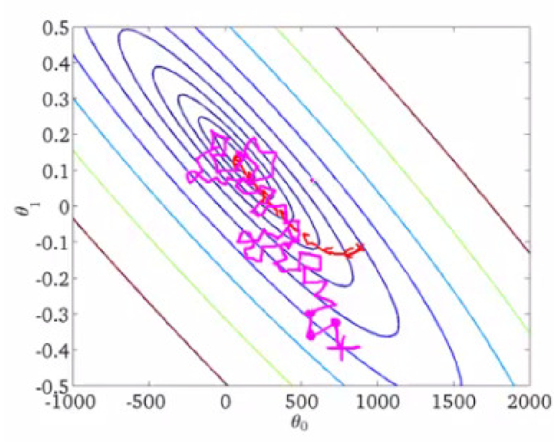


http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \alpha^{(k)} \nabla \Lambda \left(\vec{w}^{(k)} \right)$$

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

SGD



http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \beta^{(k)} \nabla f \left(\vec{w}^{(k)}, \vec{x}_{\eta^{(k)}}; y_{\eta^{(k)}} \right)$$

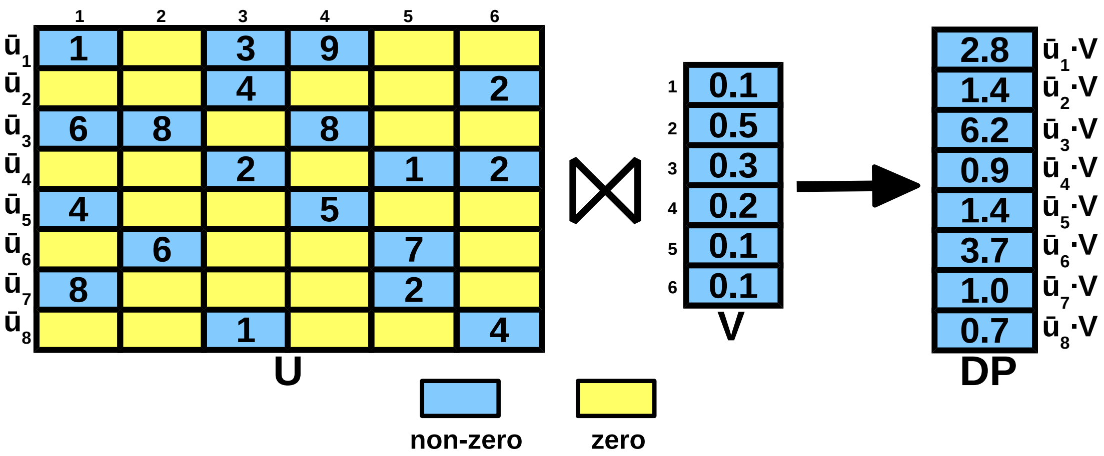
- Approximate gradient at data point
- One step for each random data point
- Faster convergence far from minimum

Agenda

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

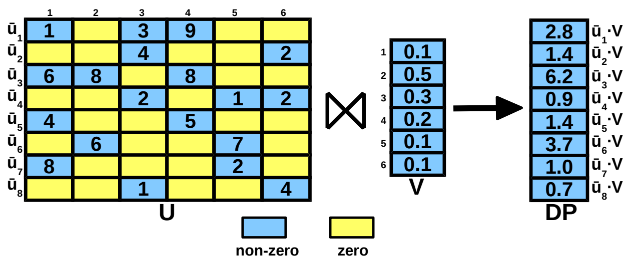
The Big Model Dot-Product Problem

- 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

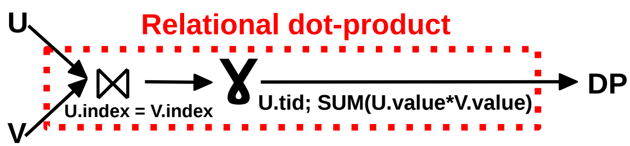
- Store sparse matrix U and dense vector V as two relational tables
- `U(index INTEGER, value NUMERIC, tid INTEGER)`
  - $\{(1,1,1); (3,3,1); (4,9,1); \ldots\}$
- `V(index INTEGER, value NUMERIC)`
  - $\{(1,0.1); (2,0.5); (3,0.3); (4,0.2); (5,0.1); (6,0.1)\}$



- SQL query to compute *DP* **is blocking**

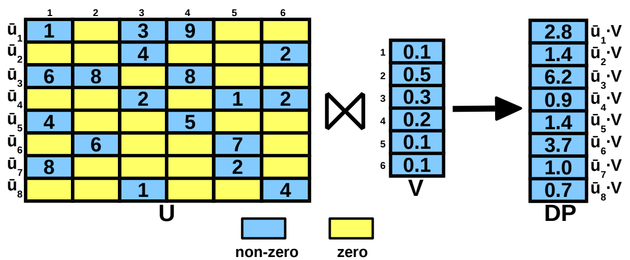
```
SELECT U.tid, SUM(U.value*V.value)
FROM U, V
WHERE U.index=V.index
GROUP BY U.tid
```

- Vectors are represented as tuples which incurs redundancy



Array-Relation Join

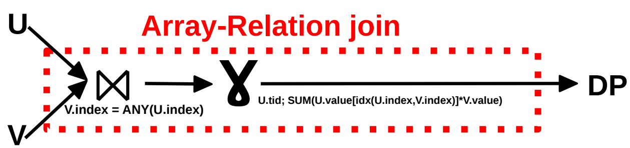
- Store sparse matrix U in the coordinate representation with two `ARRAY` attributes for the non-zero index and the corresponding value
- `U(index INTEGER[], value NUMERIC[], tid INTEGER)`
  - $\{([1,3,4], [1,3,9], 1); \ldots\}$
- `V(index INTEGER, value NUMERIC)`
  - $\{(1,0.1); (2,0.5); (3,0.3); (4,0.2); (5,0.1); (6,0.1)\}$



- SQL query to compute *DP* **is blocking**

```
SELECT U.tid,
 SUM(U.value[idx(U.index,V.index)]*
     V.value)
FROM U, V
WHERE V.index = ANY(U.index)
GROUP BY U.tid
```

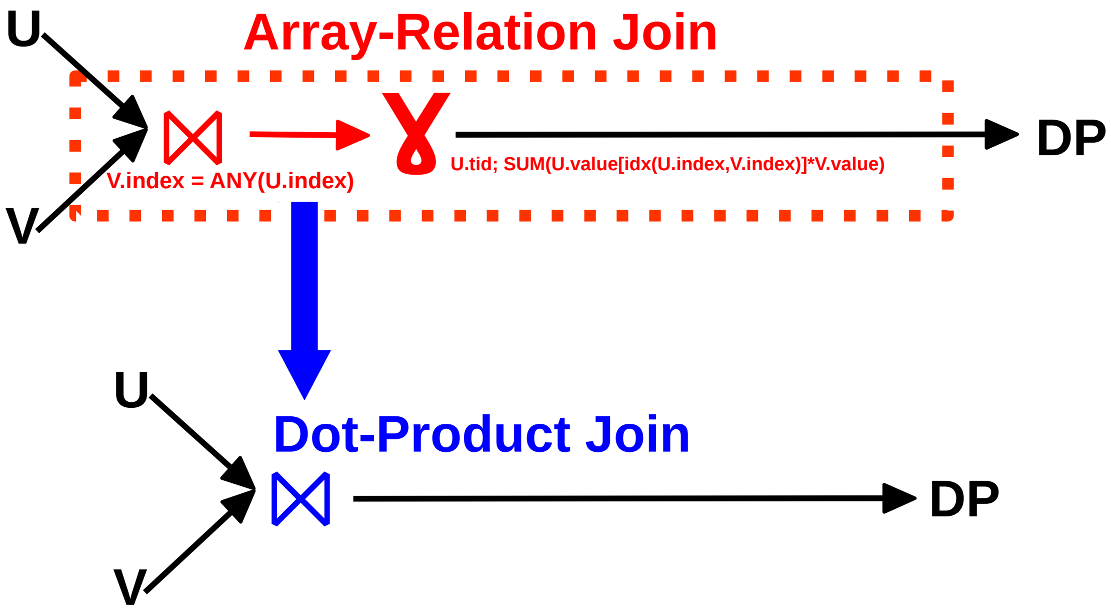
- Intermediate join size can be very large



Agenda

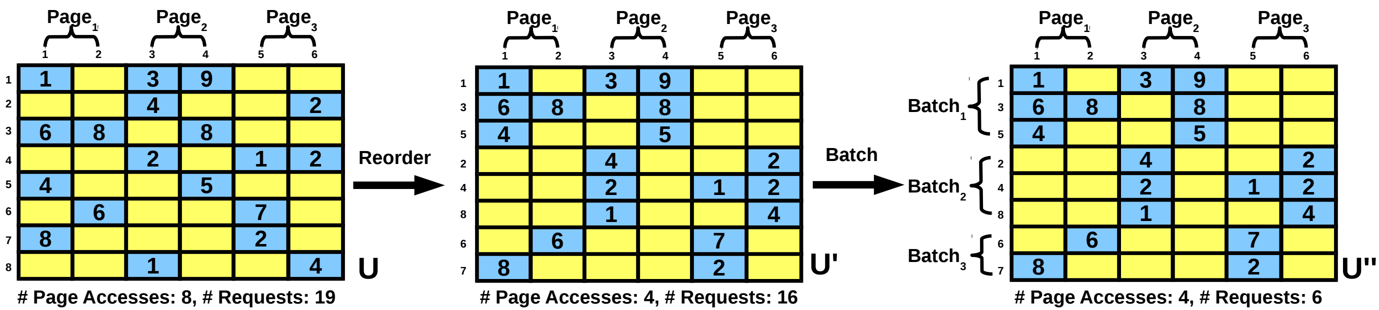
- Big Model Analytics
- Gradient Descent Optimization
- Big Model Dot-Product
- **Dot-Product Join Operator**
  - Vector Reordering
  - Batch Execution
  - Gradient Descent Integration
- Conclusions

Dot-Product Join Operator – Idea



- 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

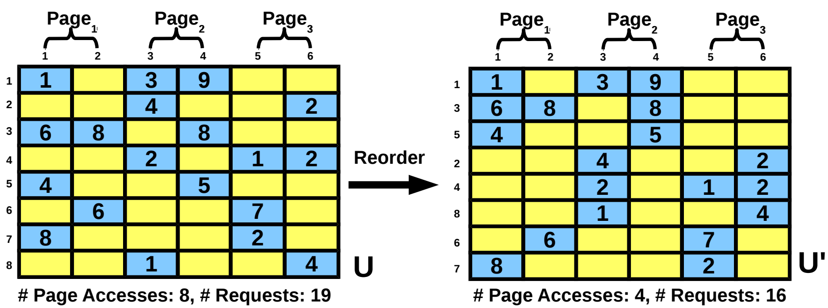
Require:
U (index INTEGER[],value NUMERIC[],tid INTEGER)
V (index INTEGER,value NUMERIC)
memory budget M

Ensure: DP (tid INTEGER,product NUMERIC)

for each page $u_p \in U$ **do**
<u>OPTIMIZATION</u>
Reorder vectors $\vec{u}_i$ to cluster similar vectors together
Group vectors $\vec{u}_i$ into batches B_j that access at most M pages from V
<u>BATCH EXECUTION</u>
for each batch B_j **do**
Collect pages $v_p \in V$ accessed by vectors $\vec{u}_i \in B_j$ into a set $v_{B_j} = \{v_p \in V | \exists \vec{u}_i \in B_j$ that accesses $v_p\}$
Request access to pages in v_{B_j}
<u>DOT-PRODUCT COMPUTATION</u>
$dp_i \leftarrow$ *Dot-Product*$(\vec{u}_i$,V)
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



K-Center Clustering

Require: Vectors $\{\vec{u}_1, \ldots, \vec{u}_N\}$ with page requests
Ensure: Reordered input vectors $\{\vec{u}_{i_1}, \ldots, \vec{u}_{i_N}\}$
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: $\mathcal{O}(kdN)$; 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 $\{\vec{u}_{i_1}, \ldots, \vec{u}_{i_N}\}$

Compute LSH tables

LSH-based nearest neighbor search

```
Initialize $\vec{u}_{i_1}$ with a random vector $\vec{u}_i$
for $j = 1$ to $N - 1$ do
    Let $X_k$ be the set of vectors co-located in the same bucket with $\vec{u}_{i_j}$ in hash table
    $Hash_k$ and not selected
    $X \leftarrow X_1 \cup \cdots \cup X_b$
    Let $\vec{u}_{i_{j+1}}$ be the vector in $X$ with the minimum set difference cardinality $\left|C^{i_{j+1}, i_j}\right|$ to
    the current vector $\vec{u}_{i_j}$
end for
```

- Complexity: $\mathcal{O}(mdN)$; 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

Compute page request frequency across vectors $\{\vec{u}_1, \ldots, \vec{u}_N\}$
for each vector $\vec{u}_i$ **do**
  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
  Reorder the bitset in decreasing order of the page request frequency, i.e., index 1 corresponds to the most frequent page
end for

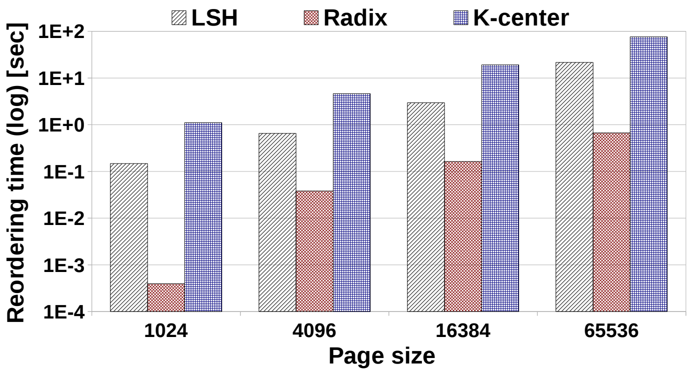
Radix sort

Apply radix sort to the set of bitsets
Let $\vec{u}_{i_j}$ be the vector corresponding to the bitset at position j in the sorted order

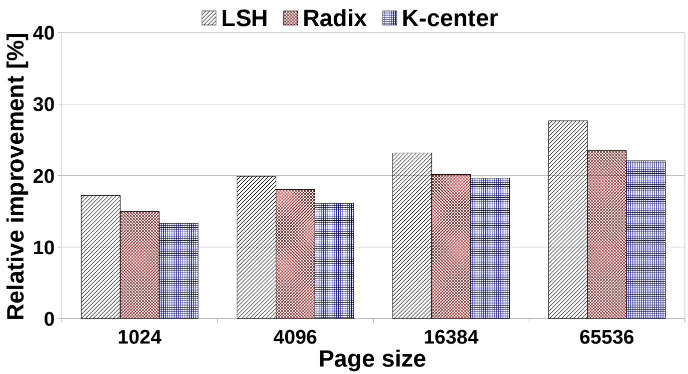
- Complexity: $\mathcal{O}(dN)$; N d-dimensional vectors
- Strict ordering between two vectors based on a single dimension at a time

Heuristics Experimental Comparison

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



Average reordering time per page

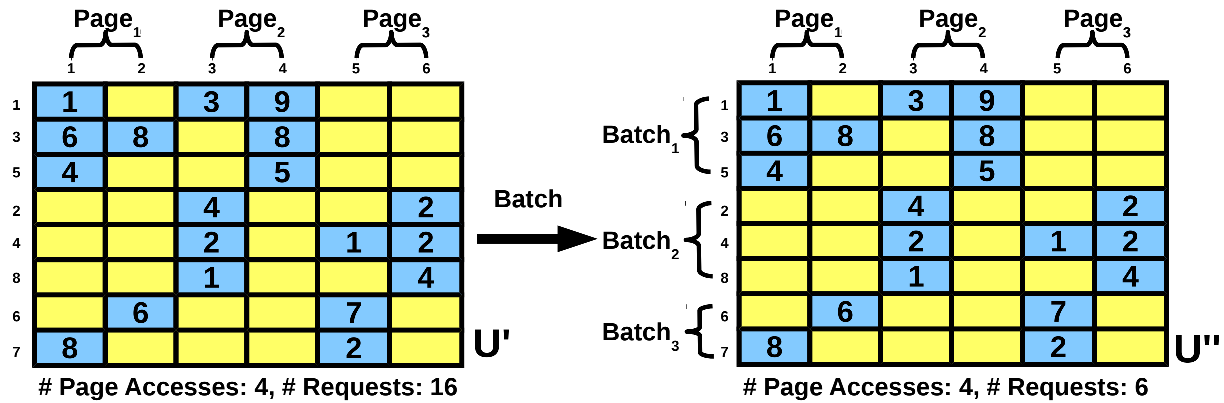


Relative improvement over LRU

- 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

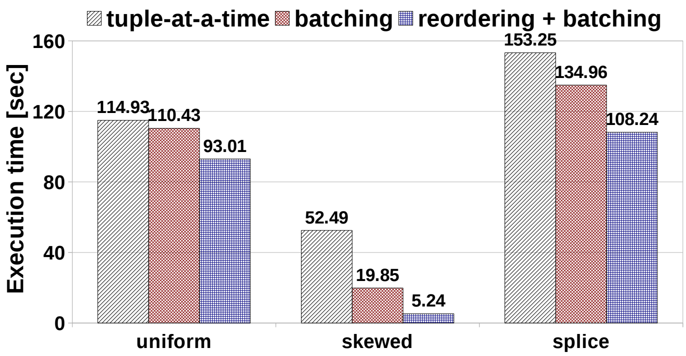
- Minimize number of requests for cells in V
- Construct batches by iterating over reordered vectors until memory budget for pages in V is full (optimal)
- Page accesses from same batch are grouped into a single request
- Enhanced buffer manager with page set requests



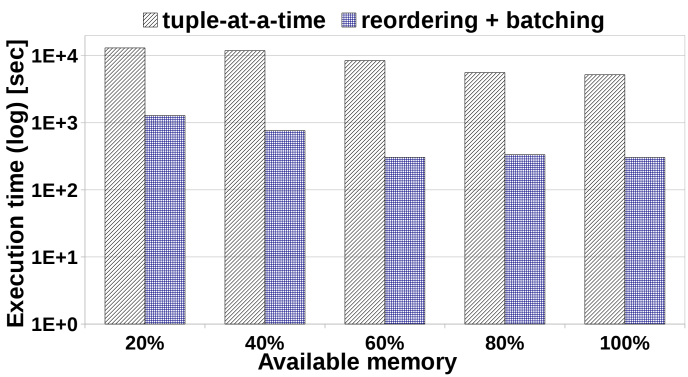
Batch Execution Evaluation

Dataset	# Dims	# Examples	Size	Model
`uniform`	1B	80K	4.2 GB	8 GB
`skewed`	1B	1M	4.5 GB	8 GB
`splice`	13M	500K	30 GB	100 MB

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

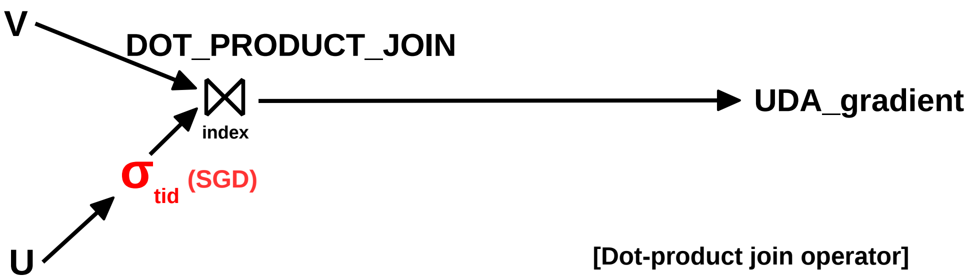


Average dot-product join time per page

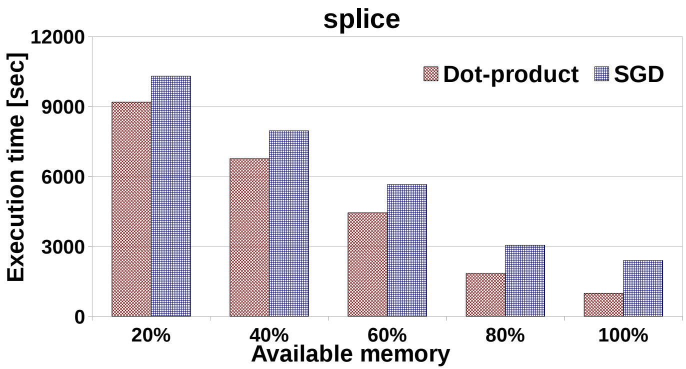


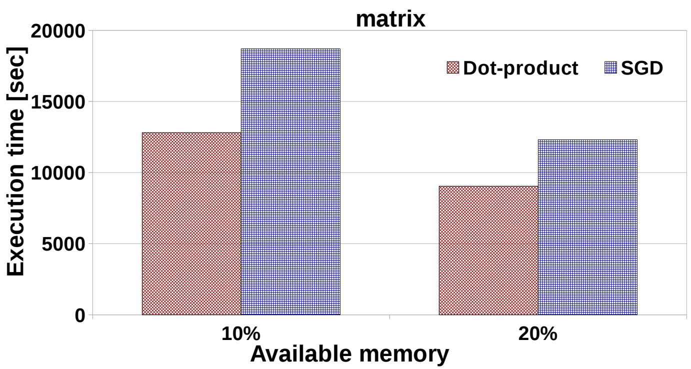
Dot-product join time over `skewed`

Gradient Descent Integration



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





Dot-Product Join vs. Alternatives

Dataset	# Dims	# Examples	Size	Model
`uniform`	1B	80K	4.2 GB	8 GB
`skewed`	1B	1M	4.5 GB	8 GB
`matrix`	10M x 10K	300M	4.5 GB	80 GB
`splice`	13M	500K	30 GB	100 MB
`MovieLens`	6K x 4K	1M	24 MB	80 MB

- 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

	GLADE R	PG R	PG A	MonetDB R	SciDB A	DOT-PRODUCT JOIN
`uniform`	37851	N/A	N/A	4280	N/A	**3914**
`skewed`	13000	N/A	N/A	4952	N/A	**786**
`matrix`	N/A	N/A	N/A	N/A	N/A	**9045**
`splice`	9554	81054	N/A	**2947**	N/A	6769
`MovieLens`	905	5648	72480	N/A	1116	**477**

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 ???