ColumnML: Column-Store Machine Learning with On-The-Fly Data Transformation

Presented by Minghao Li and Weifan Jiang
Online Analytical Processing Queries (OLAP)

```
SELECT SUM, MIN, MAX,...
FROM my_table
WHERE condition_1 AND condition 2
GROUP BY ...;
```
Online Analytical Processing Queries (OLAP)

```
SELECT SUM, MIN, MAX,...
FROM my_table
WHERE condition_1 AND condition 2
GROUP BY ...;
```

What if we want:
- Causality analysis?
- Regression?
- Classification
Integrate ML with OLAP

```
SELECT SUM, MIN, MAX,...
FROM my_table
WHERE condition_1 AND condition_2
GROUP BY ...;
```
Why ML+DMBS integration?

Reduce data movement?

Security?

Development Pipeline?

Why is ML+OLAP hard?

- Non-disrupted
- Efficiency

```sql
SELECT SUM, MIN, MAX,...
FROM my_table
WHERE condition_1 AND condition 2
GROUP BY ...;
```
Generalized Linear Models (G.L.M.)

\[
\langle x, a \rangle = \sum_{i=1}^{n} x_i a_i
\]

input as n-dim vectors

model as n-dim vector

inference using dot product
Training G.L.M.s as an optimization problem

\[
\min_{x \in \mathbb{R}^n} \left( \frac{1}{m} \sum_{i=1}^{m} J(\langle x, a_i \rangle, b_i) + \lambda \| x \|_1 \right)
\]

- Avg. from all training samples
- model weight
- features
- labels
- choice of loss function: penalize difference
- regularization
Challenge 1: Column-oriented data layout

Image courtesy: https://www.heavy.ai/technical-glossary/columnar-database
Challenge 1: Column-oriented data layout

Efficient column-wise access

Image courtesy: https://www.heavy.ai/technical-glossary/columnar-database
Challenge 1: Column-oriented data layout

Image courtesy: https://www.heavy.ai/technical-glossary/columnar-database
Stochastic gradient descent

\[ x = \text{random initialization} \]

\[ \text{for epoch } = 1, 2, 3 \ldots \]

\[ \text{for randomly selected samples } (a_i, b_i): \]

\[ x = x - \eta \cdot \nabla J(x, a_i, b_i) \]
Better: Stochastic Coordinate Descent

Algorithm 1: Stochastic Coordinate Descent

Initialize:
- $x = 0$, $z = 0$, step size $\alpha$
- $S(z) = \begin{cases} z & \text{for Lasso} \\ 1/(1 + \exp(-z)) & \text{for Logreg} \end{cases}$
- $T(x_j, g_j) = \begin{cases} \alpha g_j + \alpha \lambda & x_j - \alpha g_j > \alpha \lambda \\ \alpha g_j - \alpha \lambda & x_j - \alpha g_j < -\alpha \lambda \\ x_j & \text{else (to set } x_j = 0) \end{cases}$

for epoch = 1, 2, ... do
- randomly without replacement
  for $j = \text{shuffle}(1, ..., n)$ do
    $g_j = \frac{1}{m} (S(z) - b) \cdot a_{i,j}$ — partial gradient computation
    $\mu = T(x_j, g_j)$ — thresholding due to regularization
    $x_j = x_j - \mu$ — coordinate update
    $z = z - \mu a_{i,j}$ — inner-product vector update

cache inner product
Limitation: memory overhead for caching $z$

**Algorithm 1: Stochastic Coordinate Descent**

**Initialize:**
- $x = 0, z = 0$, step size $\alpha$
- $S(z) = \begin{cases} z & \text{for Lasso} \\ 1/(1 + \exp(-z)) & \text{for Logreg} \end{cases}$
- $T(x_j, g_j) = \begin{cases} \alpha g_j + \alpha \lambda & x_j - \alpha g_j > \alpha \lambda \\ \alpha g_j - \alpha \lambda & x_j - \alpha g_j < -\alpha \lambda \\ x_j & \text{else (to set } x_j = 0) \end{cases}$

**for epoch = 1, 2, ... do**
- randomly without replacement
  **for $j = shuffle(1, ..., n)$ do**
  - $g_j = \frac{1}{m} (S(z) - b) \cdot a_{i,j}$ — partial gradient computation
  - $\mu = T(x_j, g_j)$ — thresholding due to regularization
  - $x_j = x_j - \mu$ — coordinate update
  - $z = z - \mu a_{i,j}$ — inner-product vector update

num. samples $(m) \gg$ num. features $(n)$
partitioned SCD: cache-conscious

core idea: split the dataset by sample

process partitions one by one

inner product for current partition only

model aggregation every P epochs

---

**Algorithm 2: Partitioned SCD**

Initialize:
- \( x[K] = 0, x = 0 \), step size \( \alpha \)
- \( S(x) \) and \( T(x, g_j) \) as in Algorithm 1
- partition size \( m \), number of partitions \( K = m/M \)
- inner-product update period \( P \)

for epoch = 1, 2, ... do
  for \( k = 0, ..., K-1 \) (each partition) do
    for \( j = shuffle(1, ..., n) \) do
      \( \text{sub}_k = kM + 1, ..., kM + M \)
      partial gradient computation
      \( g_j = (S(\text{sub}_k) - \text{sub}_k) \cdot \text{sub}_k,j \)
      thresholding due to regularization
      \( \mu = T(x[k], g_j) \)
      coordinate update
      \( x[k] = x[k] - \mu \)
      inner-product vector update
      \( z_{\text{sub}_k} = z_{\text{sub}_k} - \mu z_{\text{sub}_k,j} \)
  end for
  end for
  global inner-product update with the averaged model

if epoch mod \( P \) then
  \( x' = (x[0] + ... + x[K-1])/K \)
  \( x' = 0 \)
  for \( k = 0, ..., K-1 \) (each partition) do
    for \( j = 1, ..., n \) do
      \( z_{\text{sub}_k} = z_{\text{sub}_k} + z_j a_{\text{sub}_k,j} \)
  end for
end if
## Hardware vs. Statistical Efficiency trade-off

<table>
<thead>
<tr>
<th></th>
<th>K (# partitions)</th>
<th>P (# epochs per model agg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smaller values</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware vs. Statistical Efficiency trade-off

<table>
<thead>
<tr>
<th></th>
<th>K (# partitions)</th>
<th>P (# epochs per model agg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger values</td>
<td>↑ Hardware</td>
<td></td>
</tr>
<tr>
<td></td>
<td>↓ Statistical</td>
<td></td>
</tr>
<tr>
<td>Smaller values</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>K (number of partitions)</td>
<td>P (# epochs per model agg.)</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Larger values</td>
<td>üp Hardware</td>
<td></td>
</tr>
<tr>
<td></td>
<td>üp Statistical</td>
<td></td>
</tr>
<tr>
<td>Smaller values</td>
<td>↓p Hardware</td>
<td></td>
</tr>
<tr>
<td></td>
<td>↓p Statistical</td>
<td></td>
</tr>
</tbody>
</table>
## Hardware vs. Statistical Efficiency trade-off

<table>
<thead>
<tr>
<th></th>
<th>K (# partitions)</th>
<th>P (# epochs per model agg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger values</td>
<td>🆆️ Hardware</td>
<td>🆆️ Hardware</td>
</tr>
<tr>
<td></td>
<td>🅵️ Statistical</td>
<td>🅵️ Statistical</td>
</tr>
<tr>
<td>Smaller values</td>
<td>🅵️ Hardware</td>
<td></td>
</tr>
<tr>
<td></td>
<td>🆆️ Statistical</td>
<td></td>
</tr>
</tbody>
</table>
## Hardware vs. Statistical Efficiency trade-off

<table>
<thead>
<tr>
<th></th>
<th>K (# partitions)</th>
<th>P (# epochs per model agg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger values</td>
<td>↑ Hardware</td>
<td>↑ Hardware</td>
</tr>
<tr>
<td></td>
<td>↓ Statistical</td>
<td>↓ Statistical</td>
</tr>
<tr>
<td>Smaller values</td>
<td>↓ Hardware</td>
<td>↓ Hardware</td>
</tr>
<tr>
<td></td>
<td>↑ Statistical</td>
<td>↑ Statistical</td>
</tr>
</tbody>
</table>
pSCD vs. SCD: Statistical Efficiency

Q: What’s the quality of the model after training X epochs using SCD or pSCD?
# pSCD vs. SCD: Statistical Efficiency

## Configuration vs. Validation Metric

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Validation Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data set: IM</strong></td>
<td></td>
</tr>
<tr>
<td>Epoch = 200</td>
<td>Log Loss</td>
</tr>
<tr>
<td>Train Size = 266k</td>
<td>Test Accuracy</td>
</tr>
<tr>
<td>Test Size = 66k</td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
</tr>
<tr>
<td><strong>Data set: AEA</strong></td>
<td></td>
</tr>
<tr>
<td>Epochs = 5k</td>
<td>Log Loss</td>
</tr>
<tr>
<td>Train Size = 32k</td>
<td>Test AUC</td>
</tr>
<tr>
<td>Test Size = 59k</td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
</tr>
<tr>
<td><strong>Data set: KDD1</strong></td>
<td></td>
</tr>
<tr>
<td>Epochs = 1k</td>
<td>Log Loss</td>
</tr>
<tr>
<td>Train Size = 391k</td>
<td>Test AUC</td>
</tr>
<tr>
<td>Test Size = 45k</td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
</tr>
<tr>
<td><strong>Data set: KDD2</strong></td>
<td></td>
</tr>
<tr>
<td>Epochs = 1k</td>
<td>Log Loss</td>
</tr>
<tr>
<td>Train Size = 131k</td>
<td>Test AUC</td>
</tr>
<tr>
<td>Test Size = 45k</td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
</tr>
</tbody>
</table>
## pSCD vs. SCD: Statistical Efficiency

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Validation Metric</th>
<th>SCD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data set: IM</strong></td>
<td>Log Loss</td>
<td>0.10154</td>
</tr>
<tr>
<td>Epoch = 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Size = 266k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Size = 60k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data set: AEA</strong></td>
<td>Log Loss</td>
<td>0.13531</td>
</tr>
<tr>
<td>Epochs = 5k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Size = 32k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Size = 59k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data set: KDD1</strong></td>
<td>Log Loss</td>
<td>0.24672</td>
</tr>
<tr>
<td>Epochs = 1k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Size = 391k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Size = 45k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data set: KDD2</strong></td>
<td>Log Loss</td>
<td>0.32285</td>
</tr>
<tr>
<td>Epochs = 1k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Size = 131k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Size = 45k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition Size = 16k</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# pSCD vs. SCD: Statistical Efficiency

## Table of Results

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Validation Metric</th>
<th>SCD</th>
<th>pSCD $P=\infty$</th>
<th>pSCD $P=100$</th>
<th>pSCD $P=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data set: IM</strong></td>
<td>Log Loss</td>
<td>0.10154</td>
<td>0.10575</td>
<td>+0.15%</td>
<td>+0.36%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.10380</td>
<td>+2.23%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Accuracy</td>
<td>96.17%</td>
<td>96.071%</td>
<td>-0.10%</td>
<td>96.196%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>96.109%</td>
<td>-0.06%</td>
<td></td>
</tr>
<tr>
<td><strong>Data set: AEA</strong></td>
<td>Log Loss</td>
<td>0.13531</td>
<td>0.25927</td>
<td>+91.61%</td>
<td>0.13947</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.14972</td>
<td>+10.65%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test AUC</td>
<td>0.91029</td>
<td>0.86880</td>
<td>-4.56%</td>
<td>0.91013</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.90891</td>
<td>-0.15%</td>
<td></td>
</tr>
<tr>
<td><strong>Data set: KDD1</strong></td>
<td>Log Loss</td>
<td>0.24672</td>
<td>0.24712</td>
<td>+0.16%</td>
<td>0.24698</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.24701</td>
<td>+0.12%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test AUC</td>
<td>0.62430</td>
<td>0.62369</td>
<td>-0.10%</td>
<td>0.62226</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.62274</td>
<td>-0.25%</td>
<td></td>
</tr>
<tr>
<td><strong>Data set: KDD2</strong></td>
<td>Log Loss</td>
<td>0.32285</td>
<td>0.32294</td>
<td>+0.03%</td>
<td>0.32285</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.32286</td>
<td>+0.003%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test AUC</td>
<td>0.61145</td>
<td>0.61144</td>
<td>-0.002%</td>
<td>0.61139</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.61145</td>
<td>0%</td>
<td></td>
</tr>
</tbody>
</table>

Comparable results for $P = 10$!
pSCD vs. SCD: Hardware Efficiency

Q: How fast is one epoch using SCD/pSCD under various hardware configurations?
pSCD vs. SCD: Hardware Efficiency

![Graph showing processing rate comparison between SCD and pSCD for various AVX configurations. The graph indicates that pSCD outperforms SCD, with higher processing rates.](image)
pSCD vs. SCD: Hardware Efficiency

- pSCD has larger throughput!
- pSDC utilizes hardware intrinsics & multithreading better!
pSCD vs. SCD: Hardware Efficiency

![Comparison Graphs]

Better

Same observation for other learning tasks!
pSCD/SCD vs. SGD: Hardware Efficiency

Comparison done on single-threaded CPU because SGD is hard to parallelize.
pSCD/SCD vs. SGD: Hardware Efficiency

Comparison done on single-threaded CPU because SGD is hard to parallelize.

SGD efficiency drops from row to column stored DBMS!
pSCD/SCD vs. SGD: Hardware Efficiency

Comparison done on single-threaded CPU because SGD is hard to parallelize.

SGD efficiency increases with batch size in column-stores!
pSCD/SCD vs. SGD: Hardware Efficiency

Comparison done on single-threaded CPU because SGD is hard to parallelize.

SGD with large batch size comparable with pSCD!
pSCD/SCD vs. SGD: overall comparison on CPU

Q: How does model’s loss change with time for SGD and pSCD?

Ideal: reduce loss in short time
pSCD/SCD vs. SGD: overall comparison on CPU

(a) Convergence over time for KDD1.
For row-stores, SGD is clearly the best.
pSCD/SCD vs. SGD: overall comparison on CPU

For column-stores, SGD with large batch size is comparable with pSCD and SCD.

For row-stores, SGD is clearly the best.
pSCD/SCD vs. SGD: overall comparison on CPU

Same observation for other learning tasks!
pSCD/SCD vs. SGD: overall comparison on CPU

Q: Why would we prefer SCD/pSCD over SGD, or the other way around?

Same observation for other learning tasks!
Challenge 2: ML-unfriendly underlying data layout

In DBMS, data is usually stored with compression and encryption. ML algorithms **DO NOT** work with compressed/encrypted data by default -> on-the-fly data transformation
Challenge 2: ML-unfriendly underlying data layout

In DBMS, data is usually stored with compression and encryption. ML algorithms DO NOT work with compressed/encrypted data by default -> on-the-fly data transformation

Data transformation is bottleneck using CPU.
Solution: accelerate with FPGA
Key idea: parallel memory I/O & computation
Key idea: parallel memory I/O & computation

Write-back Engine  Fetch Engine  D.P. 1

Compute Engine

Data Partition 2
Data Partition 3
...
Data Partition K
Key idea 1: parallel memory I/O & computation
Key idea: parallel memory I/O & computation
Key idea: parallel memory I/O & computation
Key idea: parallel memory I/O & computation

- Write-back Engine
- Fetch Engine
- Compute Engine
- SCD’ing on D.P. 1
- D.P. 2
- Data Partition 3
- ...
Key idea: parallel memory I/O & computation

- Write-back Engine
- Fetch Engine
- D.P. 2
- Compute Engine
- finished D.P. 1

- Data Partition 3
- Data Partition K
Key idea: parallel memory I/O & computation
Why pSCD is more suitable for FPGA than SGD?

Model kept on FPGA on-chip memory for each update. (otherwise waste memory bandwidth)
Why pSCD is more suitable for FPGA than SGD?

n: number of features
K: number of partitions
pSCD's per-partition model requires \( \frac{n}{K} \) amount of FPGA memory!
Why pSCD is more suitable for FPGA than SGD?

SGD: One row is used twice per update: initial dot product & partial gradient computation.

However, SGD requires large batch size for comparable hardware utilization with pSCD.
Multiple SCD engines with load balancing

High memory-access latency results in non-saturated FPGA memory bandwidth.
Multiple SCD engines with load balancing

High memory-access latency results in non-saturated FPGA memory bandwidth.

Running multiple SCD Engines concurrently.
Evaluation

Comparison done on a multi-core CPU and a FPGA. Delta-encoding decompression; AES256-CBC decryption.

![Processing Rate (GB/s)](image)

(a) Lasso, SYN1
(b) Lasso, SYN2
(c) Logreg, IM
Evaluation

For SCD, CPU performs better due to higher memory bandwidth than FPGAs.

Comparison done on a multi-core CPU and a FPGA. Delta-encoding decompression; AES256-CBC decryption.
Evaluation

For data with compression, encryption, or both, using multi-SCD-engine FPGA + pSCD gives best hardware utilization!
Effect of compression rate

(a) Compression rate analysis. For pSCD $P = 10$. 
Effect of compression rate

Better efficiency $\uparrow$ with higher compression rate $\uparrow$

(a) Compression rate analysis. For pSCD $P = 10$. 

- $\uparrow$ decompression cost
- $\downarrow$ memory I/O per partition
Effect of compression rate

Better efficiency with higher compression rate

Longer decompression time compensated by reduced memory I/O latency.

(a) Compression rate analysis. For pSCD $P = 10$. 

↑ decompression cost
↓ memory I/O per partition
Effect of Global Update Frequency ($P$)

Higher $P$ $\uparrow$ gives higher hardware utilization $\uparrow$.

One global update per $P$ epochs $\downarrow$ statistical utility.
Effect of Global Update Frequency (P)

Higher P ↑ gives higher hardware utilization ↑.

Previously: good model quality for up to P = 10 (within 3.07% of SCD)

One global update per P epochs ↓ statistical utility
Conclusion

Goal: Seamlessly integrate ML to Column-oriented DBMS

Challenge 1: ML algorithms with row-wise access pattern is inefficient
Solution: use Stochastic Coordinate Descent for column-wise access pattern; use partitioned SCD for cache consciousness.

Challenge 2: DBMS stores data with encryption/compression, on-the-fly transformation for ML is expensive
Solution: Use FPGA to parallelize memory I/O + transformation time and SCD computation time.

User can pick system configurations based on knowledge of underlying data layout!
Discussion

Other limitations to FPGA-based solution?

Self-designed ML in OLAP system?