Secure Parallel Computation on National-Scale Volumes of Data

Sahar Mazloom, Phi Hung Le, Samuel Ranellucci, S. Dov Gordon
Learning on User Data
“Computation on Encrypted Data”

Secure Multi-Party Computation:

Goal: Creating methods for parties to jointly compute a function over their inputs while keeping those inputs private.
Example of a ML algorithm

(Sparse) Matrix Factorization:
• De-composition of a sparse matrix of Ratings ($R$), into Users’ ($U$), and Items’ ($V$) matrices.

Gradient Descend:
• An optimization algorithm used in many machine learning algorithms.
Matrix Factorization using Gradient Descent for Movie Recommendation

Objective function:  
\[ L = \min \sum_{(i,j) \in M} (r_{ij} - \langle u_i, v_j \rangle)^2 + \mu \sum_{i \in \mathcal{I}} \| u_i \|^2 + \lambda \sum_{j \in \mathcal{J}} \| v_j \|^2 \]

\[ \begin{cases} 
  \text{User gradient: } \delta_{u_i} = -2 \sum_{j \in \mathcal{J}} v_j (r_{ij} - \langle u_i, v_j \rangle) + 2\mu u_i \\
  \text{Movie gradient: } \delta_{v_j} = -2 \sum_{i \in \mathcal{I}} u_i (r_{ij} - \langle u_i, v_j \rangle) + 2\lambda v_j 
\end{cases} \]
Distributed Graph Parallel Computation

**Non-secure Frameworks:**
MapReduce, GraphLab, PowerGraph [Gonzalez et al. 2012]

**Supported algorithms:**
Matrix Factorization, Histogram, PageRank, Markov Random Field, Parameter Learning, Name Entity Resolution, ...
PowerGraph: Think as a Vertex!
Move computation to data

GAS model of Operation

**Gather (Reduce)**
Accumulate information about neighborhood

*User Defined:*
- Gather(\(\bullet\)) \(\rightarrow\) \(\Sigma\)
- \(\Sigma_1 \oplus \Sigma_2 \rightarrow \Sigma_3\)

**Apply**
Apply the accumulated value to center vertex

*User Defined:*
- Apply(\(\mathbb{Y}\), \(\Sigma\)) \(\rightarrow\) \(\mathbb{Y}'\)

**Scatter**
Update adjacent edges and vertices.

*User Defined:*
- Scatter(\(\mathbb{Y}\)) \(\rightarrow\)

Update Edge Data & Activate Neighbors
Secure Graph Parallel Computation

- **GraphSC** [Nayak et al. SP’15]

- Use **Oblivious Sort** to hide *node degree* and *edge structure*

Complexity:

\[ O((|E| + |V|) \log^2 (|E| + |V|)) \]

Running time:

6K users, 4K movies, 1 M Ratings => **13 Hrs**

Threat model: Honest-But-Curious Adversary
Primary Question [Mazloom, Gordon CCS’18]

Can we make secure computation algorithms faster if we allow *something small* to be learned?
And prove the leakage is *Differentially Private!*
Differentially-Oblivious Graph Parallel Computation

- **OblivGraph** [Mazloom, Gordon CCS’18]

- Noisy node degree by adding dummy edges

- No. of dummy edges determined by *DP parameters*

- Use **Oblivious Sort** to hide the *edge structure*

Users

<table>
<thead>
<tr>
<th>Users</th>
<th>Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shuffle

Complexity:

\[
O((|E'| + \alpha|V|) \log(|E'| + \alpha|V|)), \quad \alpha: O\left(\frac{\log \delta - \log m}{\varepsilon}\right)
\]

V vertices, E’ all edges
OblivGraph [Mazloom, Gordon CCS’18]
OblivGraph [Mazloom, Gordon CCS’18]

Running time:
6K users, 4K movies, 1 M Ratings => 2 Hrs

Threat model: Honest-But-Curious Adversary
Can we make these *differentially private secure computation* algorithms even faster?
Can we do better?

- Low communication MPC  [Gordon et al. Asiacrypt’18]
- Differentially Private Leakage in Secure Computation [Mazloom, Gordon CCS’18]
- Graph Parallel Computation
  => Constructing an MPC protocol that can

Running time:

  6K users, 4K movies, 1 M Ratings => 25 Sec

  MF on 20 million inputs < 6 mins (MovieLens dataset)

Histograms on 300 million inputs in only 4 mins  (Counting users in each zip code)
Key playing factors

- Using 4 computation servers instead of 2
- Linear Oblivious Shuffle instead of Quasi-Linear OblivShuffle
- Fixed-Point Arithmetic Computation instead of Boolean Circuit
- Secure against one malicious adversary
Challenge 1

- The party that access the data should **NOT** learn the shuffling pattern

Solution

- Partition the tasks between 4 parties:
  - Group 1: **Shuffle** the data
  - Group 2: **Access** the data
Challenge 2

Solution

- Secure against active adversary
- Verifying the result of each operation to detect cheating behavior
Malicious-secure 4-party Secure Parallel Computation

Alice → OblivShuffle → Verify Gather → OblivApply → Verify Scatter → Cross-Check → OblivScatter → OblivApply → OblivGather → Verify Shuffle → Bob

Improved by 230X

Improved by 880X
Challenge 3  
- Fixed-Point Arithmetic Computation

Solution  
- Truncation and handling the rounding error
Cross–Check Verification after Apply phase inspired by [Gordon et al. 2018]

Alice & Bob compute on some masked values and truncate

Charlotte & David compute on some masked values and truncate

If their results don’t match?

Abort

The verification may **Fail** if data is a decimal value, and it’s NOT because of malicious behavior!
Implementation Results

Implemented in C++, run experiments on AWS

Multiple benchmark algorithms, including Matrix Factorization and Histogram

4 computation servers, 32 cores each, 10 Gbps network

<table>
<thead>
<tr>
<th>Edges</th>
<th>Users</th>
<th>Items</th>
<th>$\varepsilon$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>6K</td>
<td>4K</td>
<td>0.3</td>
<td>$2^{-40}$</td>
</tr>
<tr>
<td>10M</td>
<td>72K</td>
<td>10K</td>
<td>0.3</td>
<td>$2^{-40}$</td>
</tr>
<tr>
<td>20M</td>
<td>138K</td>
<td>27K</td>
<td>0.3</td>
<td>$2^{-40}$</td>
</tr>
<tr>
<td>300M</td>
<td>300M</td>
<td>42K</td>
<td>0.3</td>
<td>$2^{-40}$</td>
</tr>
</tbody>
</table>

Input size and privacy parameters for different experiments
Run Time on National-Scale Histogram Problem

<table>
<thead>
<tr>
<th>Processors / Edges</th>
<th>1M</th>
<th>10M</th>
<th>20M</th>
<th>300M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.8</td>
<td>85.0</td>
<td>207.7</td>
<td>2149.4</td>
</tr>
<tr>
<td>2</td>
<td>7.5</td>
<td>46.5</td>
<td>98.1</td>
<td>1136.5</td>
</tr>
<tr>
<td>4</td>
<td>4.3</td>
<td>28.0</td>
<td>57.78</td>
<td>643.2</td>
</tr>
<tr>
<td>8</td>
<td>2.7</td>
<td>16.2</td>
<td>34.39</td>
<td>382.5</td>
</tr>
<tr>
<td>16</td>
<td>1.8</td>
<td>11.2</td>
<td>23.3</td>
<td>279.2</td>
</tr>
<tr>
<td>32</td>
<td>1.5</td>
<td>10.1</td>
<td>21.67</td>
<td><strong>250.4</strong></td>
</tr>
</tbody>
</table>

Run time (s) for computing Histogram problem on different input sizes (LAN)
Counting people in each zip code
Run Time Large-Scale MF Problem

<table>
<thead>
<tr>
<th>Processors / Edges</th>
<th>1M</th>
<th>10M</th>
<th>20M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>258.3</td>
<td>1639.7</td>
<td>3401.8</td>
</tr>
<tr>
<td>2</td>
<td>132.9</td>
<td>834.7</td>
<td>1913.7</td>
</tr>
<tr>
<td>4</td>
<td>80.4</td>
<td>455.57</td>
<td>1055.95</td>
</tr>
<tr>
<td>8</td>
<td>44.6</td>
<td>292.2</td>
<td>613.1</td>
</tr>
<tr>
<td>16</td>
<td>28.2</td>
<td>190.6</td>
<td>423.7</td>
</tr>
<tr>
<td>32</td>
<td>25.1</td>
<td>163.4</td>
<td><strong>357.2</strong></td>
</tr>
</tbody>
</table>

Run time (s) for computing Matrix Factorization problem on real-world dataset, *MovieLens* on different input sizes for Movie Recommendation.
Run Time Comparison with previous works

<table>
<thead>
<tr>
<th></th>
<th>GraphSC</th>
<th>OblivGraph</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>13hrs</td>
<td>2hrs</td>
<td>25s</td>
</tr>
</tbody>
</table>

Run time comparison on this work vs. OblivGraph vs. GraphSC. Single iteration of Matrix Factorization on real-world dataset, MovieLens with 6K users ranked 4K movies with 1M ratings.
Summarize

Goal:

Learning on large-scale data with security and privacy

• Secure MPC for Privacy Preserving Machine Learning
• Secure against one malicious corruption
• Leverage Differential Privacy to improve efficiency
Secure Parallel Computation on National-Scale Volumes of Data

Sahar Mazloom, Phi Hung Le, Samuel Ranellucci, S. Dov Gordon
sseyma@gmu.edu
Code is publicly available!

Thanks!

Q&A