Oblivious Coopetitive Analytics Using Hardware Enclaves

Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica (UC Berkeley)

EuroSys 2020
April 28, 2020
The need for coopetitive analytics

• Analytics can extract value from big data
• But datasets often span multiple competing parties
Example: Financial risk assessment

- Banks want to assess systemic risk
- This requires cooperation among competing banks
- Sharing data creates security, regulatory, business, and liability concerns

"How much subprime debt have all banks issued?"

```
SELECT SUM(loan_amount)
FROM customer c
JOIN loan l ON c.ssn = l.ssn
WHERE credit_score < 630;
```
Threat model

- **Network attacker** can see and modify all network traffic but cannot access machines.

- **Malicious party attackers** can additionally see and modify computation within their machines + collude with other parties.

```java
if (c.credit_score < 630) {
    result[c.ssn] += c.loan_amount
}
```
Approach 1: Cryptography

**Specialized systems**: Conclave, DJoin, private intersection-sum, Prio, UnLynx, MedCo, ...

- **Limited functionality** – cannot support rich analytics

**Generic approaches**: SMCQL, AgMPC

- **Prohibitive overhead**
Approach 2: Hardware enclaves

- Trusted code runs shielded from OS and processes on the same host
- Memory access pattern leakage
Access pattern leakage

Access patterns leak information such as filter selectivity

<table>
<thead>
<tr>
<th>ID</th>
<th>Credit score</th>
<th>Loan amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>720</td>
<td>$2,500</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>$500</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>600</td>
<td>$500</td>
</tr>
</tbody>
</table>

Total loans: $1,250

Memory access

```
SELECT SUM(loan_amount)
FROM customer c
JOIN loan l ON c.ssn = l.ssn
WHERE credit_score < 630;
```
Oblivious algorithms

Oblivious algorithms hide access patterns at a performance cost

<table>
<thead>
<tr>
<th>ID</th>
<th>Credit score</th>
<th>Loan amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>720</td>
<td>$2,500</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>$500</td>
</tr>
<tr>
<td>2</td>
<td>600</td>
<td>$250</td>
</tr>
<tr>
<td>3</td>
<td>600</td>
<td>$500</td>
</tr>
</tbody>
</table>

```
SELECT SUM(loan_amount)
FROM customer c
  JOIN loan l ON c.ssn = l.ssn
WHERE credit_score < 630;
```
Previous approaches using hardware enclaves

Not oblivious: SONE, Graphene, Haven, VC3

- Side channel leakage

Oblivious: Cipherbase, Opaque

- Must maintain remote copy of large datasets; expensive to update
- If applied to WAN setting, inefficient due to high-bandwidth shuffles
Oblivious Coopetitive Queries (OCQ)

• Designed for oblivious coopetitive analytics
• Supports general SQL queries with better performance than previous approaches
• Protects against network attacker and malicious party attackers (in the hardware enclave model)
Oblivious Coopetitive Queries (OCQ)

Parties must agree on fixed queries and input data in advance

Replicated across parties

Authenticated operators on parties’ own data

Oblivous operators on joint data

Each party must have at least one hardware enclave
Challenges and Techniques

1. Combining data of mixed sensitivities
   → Approach: Mixed-sensitivity algorithms

2. Query planning with sensitive cardinalities
   → Approach: Schema-aware padding

3. Oblivious queries in the wide area
   → Federated- and security-aware planner
Sensitivity propagation

Parties specify sensitivity of each table: **Public** or **Sensitive**

Propagate sensitivity according to *foreign keys and operators*
Mixed-sensitivity oblivious join

Joining **Sensitive** tables across parties produces a **mixed-sensitivity join**

Mixed-sensitivity oblivious join algorithm:

1. Sort **Public** and **Sensitive** sides separately
2. Oblivious bitonic merge join

Up to 2.5x speedup vs. fully-oblivious join for equal-sized tables
Schema-aware padding

- Cardinalities are particularly sensitive in the federated setting
- Naïve “filter push-up” approaches to padding are very expensive
- Find tighter padding bounds using foreign key constraints

```sql
SELECT c_zip, AVG(l_amount / d_income)
FROM customer
JOIN loan ON c_ssn = l_ssn
JOIN region ON c_zip = r_zip
JOIN demographics ON r_zip = d_zip
GROUP BY c_zip
```
Federated planner

Determines how to run the query and where to run each operator

```sql
SELECT SUM(loan_amount)
FROM customer c
JOIN loan l ON c.ssn = l.ssn
WHERE credit_score < 630;
```
Evaluation setup

• 5 geo-distributed parties
• ~10 MB/s bandwidth
• Synthetic data, table sizes 4.3 MB–10 GB
OCQ vs. prior work

- Orders of magnitude faster than SMCQL and DJoin due to trusted hardware
- Faster than Opaque because OCQ can execute initial filters in plaintext
Overhead of OCQ’s security

- 2.2–25x overhead vs. insecure federated or outsourced Spark SQL
Summary of OCQ’s contributions

Efficient, general framework for oblivious coopetitive analytics
1. Mixed-sensitivity oblivious join and aggregation algorithms
2. Schema-aware padding
3. Secure coopetitive query planner