Airavat: Security and Privacy for MapReduce

Indrajit Roy, Srinath T.V. Setty, Ann Kilzer, Vitaly Shmatikov, Emmett Witchel

The University of Texas at Austin
Computing in the year 201X

- Illusion of infinite resources
- Pay only for resources used
- Quickly scale up or scale down ...
Programming model in year 201X

- Frameworks available to ease cloud programming
- **MapReduce**: Parallel processing on clusters of machines

- Data mining
- Genomic computation
- Social networks
Programming model in year 201X

- Thousands of users upload their data
  - Healthcare, shopping transactions, census, click stream
- Multiple third parties mine the data for better service

- Example: Healthcare data

- Incentive to contribute: Cheaper insurance policies, new drug research, inventory control in drugstores…

- Fear: What if someone targets my personal data?
  - Insurance company can find my illness and increase premium
Privacy in the year 201X?

• Data mining
• Genomic computation
• Social networks

Untrusted MapReduce program

Output

Information leak?
Use de-identification?

- Achieves ‘privacy’ by syntactic transformations
  - Scrubbing, k-anonymity ...
- Insecure against attackers with external information
  - Privacy fiascoes: AOL search logs, Netflix dataset

Run untrusted code on the original data?
How do we ensure privacy of the users?
Audit the untrusted code?

- Audit all MapReduce programs for correctness?

Aim: Confine the code instead of auditing

Hard to do! Enlightenment?

Also, where is the source code?
This talk: Airavat

Framework for privacy-preserving MapReduce computations with untrusted code.

Airavat is the elephant of the clouds (Indian mythology).
Airavat guarantee

Bounded information leak* about any individual data after performing a MapReduce computation.

*Differential privacy
Outline

- Motivation
- Overview
- Enforcing privacy
- Evaluation
- Summary
Background: MapReduce

\[
\text{map}(k_1,v_1) \rightarrow \text{list}(k_2,v_2) \\
\text{reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_2)
\]
MapReduce example

Map(input) → { if (input has iPad) print (iPad, 1) }
Reduce(key, list(v)) → { print (key + “,” + SUM(v)) }

Counts no. of iPads sold

Map phase: iPad, Tablet PC, iPad, Laptop
Reduce phase: (iPad, 1) → (iPad, 2)
Airavat model

- Airavat framework runs on the cloud infrastructure
  - Cloud infrastructure: Hardware + VM
  - Airavat: Modified MapReduce + DFS + JVM + SELinux
Airavat model

- Data provider uploads her data on Airavat
  - Sets up certain privacy parameters
Airavat model

- Computation provider writes data mining algorithm
  - Untrusted, possibly malicious
Airavat runs the computation, and still protects the privacy of the data providers.
Roadmap

- What is the programming model?
- How do we enforce privacy?
- What computations can be supported in Airavat?
Split MapReduce into **untrusted mapper + trusted reducer**

- Limited set of stock reducers
- No need to audit
- MapReduce program for data mining
Programming model

Need to confine the mappers!

Guarantee: Protect the privacy of data providers

MapReduce program for data mining

No need to audit
Challenge 1: Untrusted mapper

- Untrusted mapper code copies data, sends it over the network

Diagram:

- Peter
- Chris
- Meg
- Data
- Map
- Reduce
- Leaks using system resources
Challenge 2: Untrusted mapper

- Output of the computation is also an information channel

Output 1 million if Peter bought Vi*gra
Airavat mechanisms

Mandatory access control
Prevent leaks through storage channels like network connections, files…

Differential privacy
Prevent leaks through the output of the computation

Data ➔ Map ➔ Reduce ➔ Output
Back to the roadmap

- What is the programming model?
  - Untrusted mapper + Trusted reducer

- How do we enforce privacy?
  -Leaks through system resources
  -Leaks through the output

- What computations can be supported in Airavat?
Airavat confines the untrusted code

- Untrusted program
- MapReduce + DFS
- SELinux

Given by the computation provider
Add mandatory access control (MAC)
Add MAC policy

Airavat
Airavat confines the untrusted code

- We add mandatory access control to the MapReduce framework
- Label input, intermediate values, output
- Malicious code cannot leak labeled data
Airavat confines the untrusted code

- SELinux policy to enforce MAC
- Creates trusted and untrusted domains
- Processes and files are labeled to restrict interaction
- Mappers reside in untrusted domain
  - Denied network access, limited file system interaction
But access control is not enough

- Labels can prevent the output from being read
- When can we remove the labels?

```java
if (input belongs-to Peter)
    print (iPad, 1000000)
```

Output leaks the presence of Peter!
But access control is not enough

Need mechanisms to enforce that the output does not violate an individual’s privacy.
A mechanism is **differentially private** if every output is produced with similar probability whether any given input is included or not.

A mechanism is \textit{differentially private} if every output is produced with similar probability whether any given input is included or not.
A mechanism is **differentially private** if every output is produced with similar probability whether any given input is included or not

**Bounded risk for D** if she includes her data!

*Cynthia Dwork. *Differential Privacy*. ICALP 2006*
Achieving differential privacy

- A simple differentially private mechanism

- How much noise should one add?
Achieving differential privacy

- **Function sensitivity** (intuition): Maximum effect of any single input on the output
  - Aim: Need to conceal this effect to preserve privacy

- Example: Computing the average height of the people in this room has low sensitivity
  - Any single person’s height does not affect the final average by too much
  - Calculating the maximum height has high sensitivity
Achieving differential privacy

- **Function sensitivity** (intuition): Maximum effect of any single input on the output
  - Aim: Need to conceal this effect to preserve privacy

- Example: SUM over input elements drawn from $[0, M]$

\[
\begin{align*}
X_1 & \quad X_2 & \quad X_3 & \quad X_4 \\
\quad & \quad & \quad & \quad & \text{SUM} \\
\end{align*}
\]

- **Sensitivity** = $M$
- Max. effect of any input element is $M$
Achieving differential privacy

- A simple differentially private mechanism

\[ f(x) + \text{Lap}(\Delta(f)) \]

Intuition: Noise needed to mask the effect of a single input

\[ \Delta(f) = \text{sensitivity} \]

\[ \text{Lap} = \text{Laplace distribution} \]
Back to the roadmap

- What is the programming model?
  - Untrusted mapper + Trusted reducer

- How do we enforce privacy?
  -Leaks through system resources
  -Leaks through the output

- What computations can be supported in Airavat?
Enforcing differential privacy

- Mapper can be any piece of Java code ("black box") but...

- Range of mapper outputs must be declared in advance
  - Used to estimate "sensitivity" (how much does a single input influence the output?)
  - Determines how much noise is added to outputs to ensure differential privacy

- Example: Consider mapper range \([0, M]\)
  - SUM has the estimated sensitivity of \(M\)
Enforcing differential privacy

- Malicious mappers may output values outside the range.
- If a mapper produces a value outside the range, it is replaced by a value inside the range.
  - User not notified... otherwise possible information leak.

Ensures that code is not more sensitive than declared.
Enforcing sensitivity

- All mapper invocations must be independent

- Mapper may not store an input and use it later when processing another input
  - Otherwise, range-based sensitivity estimates may be incorrect

- We modify JVM to enforce mapper independence
  - Each object is assigned an invocation number
  - JVM instrumentation prevents reuse of objects from previous invocation
Roadmap. One last time

- What is the programming model?
  - Untrusted mapper + Trusted reducer

- How do we enforce privacy?
  - Leaks through system resources
  - Leaks through the output

- What computations can be supported in Airavat?
What can we compute?

- Reducers are responsible for enforcing privacy
  - Add an appropriate amount of random noise to the outputs

- Reducers must be trusted
  - Sample reducers: SUM, COUNT, THRESHOLD
  - Sufficient to perform data mining algorithms, search log processing, recommender system etc.

- With trusted mappers, more general computations are possible
  - Use exact sensitivity instead of range based estimates
Sample computations

- Many queries can be done with untrusted mappers
  - How many iPads were sold today?
  - What is the average score of male students at UT?
  - Output the frequency of security books that sold more than 25 copies today.

- ... others require trusted mapper code
  - List all items and their quantity sold

Malicious mapper can encode information in item names
Revisiting Airavat guarantees

- Allows differentially private MapReduce computations
  - Even when the code is untrusted

- Differential privacy => mathematical bound on information leak

- What is a safe bound on information leak?
  - Depends on the context, dataset
  - Not our problem
Outline

- Motivation
- Overview
- Enforcing privacy
- Evaluation
- Summary
Implementation details

**SELinux policy**
- Domains for trusted and untrusted programs
- Apply restrictions on each domain

**MapReduce**
- Modifications to support mandatory access control
- Set of trusted reducers

**JVM**
- Modifications to enforce mapper independence

450 LoC
5000 LoC
500 LoC

LoC = Lines of Code
## Evaluation: Our benchmarks

- Experiments on 100 Amazon EC2 instances
  - 1.2 GHz, 7.5 GB RAM running Fedora 8

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Privacy grouping</th>
<th>Reducer primitive</th>
<th>MapReduce operations</th>
<th>Accuracy metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOL queries</td>
<td>Users</td>
<td>THRESHOLD, SUM</td>
<td>Multiple</td>
<td>% queries released</td>
</tr>
<tr>
<td>kNN recommender</td>
<td>Individual rating</td>
<td>COUNT, SUM</td>
<td>Multiple</td>
<td>RMSE</td>
</tr>
<tr>
<td>K-Means</td>
<td>Individual points</td>
<td>COUNT, SUM</td>
<td>Multiple, till convergence</td>
<td>Intra-cluster variance</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>Individual articles</td>
<td>SUM</td>
<td>Multiple</td>
<td>Misclassification rate</td>
</tr>
</tbody>
</table>
Performance overhead

Overheads are less than 32%
Evaluation: accuracy

- Accuracy increases with decrease in privacy guarantee

- Reducer: COUNT, SUM

*Refer to the paper for remaining benchmark results*
Related work: PINQ

- Set of trusted LINQ primitives

- Airavat confines untrusted code and ensures that its outputs preserve privacy
  - PINQ requires rewriting code with trusted primitives

- Airavat provides end-to-end guarantee across the software stack
  - PINQ guarantees are language level
Airavat in brief

- Airavat is a framework for privacy preserving MapReduce computations
- Confines untrusted code
- First to integrate mandatory access control with differential privacy for end-to-end enforcement
Airavat is a framework for privacy preserving MapReduce computations
Confines untrusted code
First to integrate mandatory access control with differential privacy for end-to-end enforcement