Observing and Preventing Leakage in MapReduce

Presenter: Haozhan Sun
Motivation
Idea Mindflow
What’s MapReduce?
  - An example
How can adversary attack?
Observe and Exploit Intermediate Traffic
Two Solutions
  - Shuffle in the middle
  - Shuffle and balance
Evaluation
Discussion
My Critique
Summary
Motivation

**Existing work:**

On cloud infrastructures (e.g., AWS, Azure), for MapReduce tasks, we simply encrypt all data, access them only within secure hardware.

**The Problem:**

Still quite heavy communications between storage units (e.g., Amazon S3) → potential leakage!
The problem: Too much communication leads to info leakage. (How to prove?)

By an example: MapReduce jobs. (To show what?)

Show the adversary can infer precise info about the input by observing the communication. (Then how to solve?)

Describe two provably-secure solutions and evaluate with VC3.
What is MapReduce?

A processing technique/model that can scale data processing easily over multiple computing nodes (clusters). In other words, make Big Data computation easier.

What’s the process?

As the name shows, first Map then Reduce.

Map: Takes a set of data and converts into some key/value pairs.

Reduce: Take the output of the map above (i.e. the key/value pairs) as an input, then combine those data into smaller tuples.
An Example of Secure MapReduce

Map: Takes a set of data and converts into some key/value pairs.
Reduce: Take the output of the map above (i.e. the key/value pairs) as an input, then combine those data into smaller tuples.

Idea: Run the Map and Reduce functions in secure regions

Dashed lines: Intermediate traffic that an adversary can observe (thus can cause leakage)
Imagine: A bad guy called *Jack - the destroyer* is watching you sending Amazon parcels (actually MapReduce jobs) on some conveyors.

What he can do:

1. **System level**: Jack can’t see what’s inside the parcel, but he can record your parcel-conveying behavior between every station in the Amazon delivery system (network traffic analysis) or between every node and storage (storage traffic analysis). Then Jack knows the volume of changed parcels (data).

2. **Application level**: Jack may have background knowledge about your job. Jack knows you might be preparing for Christmas, so when he sees a quite large-size parcel, he may infer it’s a *Christmas Tree*. Then Jack knows what you are doing.

The paper mainly concerns about the intermediate traffic between *mappers* and *reducers*.
Observe and Exploit Intermediate Traffic

The authors show that an adversary can correlate some dependent intermediate traffic to, for example, lock a person (zip code, gender and DOB observed together).

The authors also show that an adversary can exploit the intermediate traffic to answer some questions (which makes us dangerous). For example:

![Distribution of Census records across age groups (left), place of birth (POB)(center) and marital status (right)]

e.g. If at some time, the adversary observes that the mapper-reducer pair A[M,R] is $A_{age}[i,'1-12'] = 1$ and $A_{POB}[i,'Africa'] = 1$, then he knows the $i$th record is:
1. In the age group ‘1-12’
2. was born in Africa.
Observe and Exploit Intermediate Traffic

Furthermore, the adversary can answer some (dangerous) questions:

Q1: What is the marital status of person #1326457 in the census?
A1: Never married. This is because $A_{MS}[1326457, 'Never Married'] = 1$.

Q2: Is there a person who aged 13-19, born in Oceania and Divorced?
A2: Yes. This is because there is exactly one index $i = 1005243$ with $A_{age}[i, '13 - 19'], A_{POB}[i, 'Oceania'], A_{MS}[i, 'Divorced']$ all equal to 1.

The person is then uniquely identified!

How to solve this?
Security Definitions

To protect the input dataset $D$, we wish MapReduce to appear data independent according to the observed traffic, and only depend on:

1. The M (mapper) and R (reducer) functions being computed.
2. The input size $|D|$.
3. The output size $|O|$.

To achieve this, two definitions are defined:

**Definition 1:**

**MapReduce Game:** This definition models that the traffic observed for any two datasets $D^0$ and $D^1$ should appear to be the same and *not reveal anything about the input data.*

**Definition 2:**

**Correlation Hiding:** This definition requires the solution to reveal *no more than* the maximum number of records that a function R has to process.

With the definitions, we are free from composition attacks which knows \{ Age: 13-19, POB: Oceania, Marital Status: Divorced \}: because we only leak how many are between 13-19, how many are born in Oceania and how many are divorced.
The Solution 1 – Shuffle in the Middle

**Goal:** Prevent intermediate traffic analysis by shuffling the key-value pairs in the middle, such that the adversary cannot trace back.

All data elements are encrypted and Mapper and Reducer code is executed inside of the secure region. The shuffling is implemented as two additional MapReduce jobs.
The Solution 1 – Shuffle in the Middle

The two algorithms (if interested):

**Algorithm 1** Melbourne Shuffle Mapper: Mapper([d_1, \ldots, d_{i+b}]) with \pi, R, max included, for example, in the binary.

1: Let d_i, \ldots, d_{i+b} be input records in a batch of size b.
2: Let \pi be the target secret permutation.
3: Let R be the number of reducers.
4: Let max be the max number of records to be sent from a mapper to a reducer.
5: for r \in \{1 \ldots R\}: bin[r] \gets []
6: for j \in \{i \ldots i + b\}:
7: id \gets (\pi(j) \mod R) + 1
8: bin[id].append((\pi(j), d_j))
9: for r \in \{1 \ldots R\} do
10: if len(bin[r]) > max abort.
11: while (len(bin[r]) < max): bin[r].append(dummy)
12: end for
13: for r \in \{1 \ldots R\} do
14: output r, bin[r]
15: end for

**Algorithm 2** Melbourne Shuffle Reducer: Reducer(r, [X|_r|])

1: Let r be this reducer’s index.
2: Let X|_r|, be input values with key r (i.e., all bins with r).
3: vals \gets []
4: for val \in X|_r| do
5: if val \neq dummy: vals.append(val) \{val is \pi(j), d_j\}
6: end for
7: Sort vals by \pi tag, strip off tags and output the result.

**Conclusion:** The Melbourne Shuffle guarantees that its network traffic for the two MapReduce jobs depend only on the size of the dataset and the number of mappers and reducers performing each job.
This solution tries to *evenly distribute the intermediate traffic sent from each mapper to each reducer*. This is done in an *offline phase* and an *online phase*.

**Offline stage:** Randomize the ordering of the input records → erases the correlations between the ordering of inputs and the values of their attributes.
Online stage: Samples the input data to collect statistics about the keys produced by mappers, in order to balance them evenly between reducers, and estimate an upper bound on the number of key-value pairs sent to reducers.

Intuitively: Use dummy padding and balanced tasks to fool the adversary.
Evaluation

The authors compare the two solutions in terms of run time. **Baseline:** the initial job on Hadoop without any protection (adopt neither of the methods)

<table>
<thead>
<tr>
<th>DataSet/Job</th>
<th>Base</th>
<th>Run time (Shuffle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Census/Age grouped</td>
<td>20</td>
<td>91 (25)</td>
</tr>
<tr>
<td>Taxi Jan/PassenN</td>
<td>39</td>
<td>122 (38)</td>
</tr>
<tr>
<td>Taxi Jan/PickupD</td>
<td>40</td>
<td>131 (43)</td>
</tr>
</tbody>
</table>

Observations:
1. Shuffling costs highly depend on the size of the data.
2. Run time significantly slower than baseline.

Run times for Shuffle-in-the-Middle.

<table>
<thead>
<tr>
<th>Attrib</th>
<th>Taxi Jan (2.5 GB)</th>
<th>Taxi Jan-Apr (10 GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$</td>
</tr>
<tr>
<td>PassenN</td>
<td>.71</td>
<td>6</td>
</tr>
<tr>
<td>PickUpD</td>
<td>.038</td>
<td>31</td>
</tr>
<tr>
<td>PassenN-1</td>
<td>.47</td>
<td>5</td>
</tr>
</tbody>
</table>

Observations:
1. Shuffle and Balance is *more efficient* than the first solution.
2. Overhead increased by 7% on average.
3. Beat the baseline in one case due to a pre-grouping operation called bin packing (so that fewer keys are returned by the mapper – lighter workload).
Discussion

1. Shuffle and Balance (2\textsuperscript{nd} solution) has a much better performance than Shuffle in the Middle (1\textsuperscript{st} solution).

2. The additional padding by Shuffle and Balance does not affect the performance as much as introducing two additional MapReduce jobs (Shuffle in the Middle)
Critiques

1. *Insufficient evaluation*. The existing evaluation seems confusing. Cannot see a strong correlation why the paper argues about the security of the proposed methods but went to analyze its run time. Seems weird why there are two well defined methods but only with half a page of evaluation.

2. *Should be more concrete examples*. The core implementation part is full of formal definition and math equations – extremely low readability. Should demo how the shuffling and padding etc. are adopted to the datasets.
Summary

1. The problem: information leakage during the heavy communication between different processing and network storage units in MapReduce jobs.

2. Then the authors teach you if you were a malicious guy, how to observe the intermediate traffic and exploit it to infer some sensitive information during a MapReduce job.

3. Then the authors propose a new data privacy definition for MapReduce jobs (because of the aforementioned leakage).

4. Finally, they introduce two secure solutions to protect the middle steps during the MapReduce (Shuffle in the Middle and Shuffle and Balance), and evaluate their performance.
Thank you for listening!

Q & A