Regular Expression Search over Encrypted Big Data in the Cloud

Mohsen Amini Salehi
Assistant Professor
CACS Department
University of Louisiana Lafayette
Research Interests

Data Sanitization in Cloud

Robust Cloud Resource Allocation for real-time processing of big data
Outline

• Background

• Security Challenges in the Cloud

• Our Solution: “RESeED”

• Evaluations

• Future Research Plans
Motivation: Growth of the Digital Universe

- 0.1 ZB in 2005
- 1 ZB in 2010
- 1.8 ZB in 2011
- 40 ZB in 2020

Source:
Motivation

• Storage Clouds have emerged in response to the data explosion

• The need for more data security on Consumer Platforms
  o Email
  o Social Networks
  o Clouds
Motivation

• Clouds are not trustworthy!
  ▪ 10% reduce in foreign contracts with Cloud providers
  ▪ Half of businesses are uncomfortable to deal with Cloud

• Solution?
  ▪ User-side Encryption!
  ▪ Clouds become dumb block storage

• Disadvantages of encryption:
  ▪ Not transparent to the user
  ▪ Involves storage and processing overheads
  ▪ No search capabilities on the stored data
• Boneh et al. provide *keyword search* over encrypted data

• What if we need more than keyword search?
  - *We are looking for all documents authored by Andrew Stuart Tanenbaum*
    - A S Tanenbaum
    - Andrew s Tanenbaum

• Boneh complexity for regular expression: $O(2^n)$
  - $(n$ number of tokens)
Problem Definition

• How can we perform a regular expression search over the encrypted data in the Cloud?

• The solution
  ▪ Should not share any information with the Cloud
  ▪ Should not require any infrastructural change from the Cloud
  ▪ Should work in multi-cloud scenarios
Scenario
Contributions

• RESeED: a solution for regular expression search over encrypted data

• Parallel RESeED

• We compare our performance results against the *grep* utility on unencrypted data.
RESeED Architecture
Extracting Meta Data

• Column Store
  - Map from Token to files in which it appeared

• Order Store
  - A fuzzy (i.e., hashed) representation of the file
  - Used to match the correct order of words in the search query
  - For each token in a file, create an N byte hash
    - What is the proper hash-width (N)?
Tuning RESeED

- Hash-width 3 bytes for order store provides a trade-off
RESeED Architecture
RESeED Search Operation

- To search for a regular expression $r$:
  1. *Convert* $r$ *into an Nondeterministic Finite Automata (NFA)*
  2. The NFA is partitioned to a set of sub-NFAs based on $\omega$-transformation
  3. For each token in Column Store:
     A. Check if it is matched with a sub-NFA
  4. For files that accept all sub-NFAs:
     A. Using Order Store, verify if they have the expression in the expected order
1. Given a regular expression, build an NFA “back(\s)?pack”
Alphabet Definition

• Alphabet of a regular expression is defined over:

\[ \Sigma = C \cup \Omega \]

• Can be divided into two disjoint subsets
  - \( C \): the set of core symbols
  - \( \Omega \): the set of separators

• Languages are composed of strings of \( C \) separated by strings of
2. The NFA is partitioned to a set of sub-NFAs based on \( \omega \)-transformation
3. Check if it is matched with a sub-NFA

- For each token in column store
  - Match against all sub-NFAs created by $\omega$-Transformation
- Results into a map from filename to satisfied sub-NFAs
RESeED Architecture
4. Using Order Store, verify if they have the expression in the expected order:

- **Determine if a file has possible satisfiability using Path NFA**
  - Walk Order Store using Path NFA
- **If Path NFA accepts, return file as a match**

Example: “back(\s)?pack”
Experimental Setup

• RFC dataset
  (6,942 txt files totaling 357 MB)

• arXiv.org dataset
  (683,620 pdf files totaling 264 GB)

• Dropbox as storage Cloud
Evaluated Benchmarks

(A) Words related to Interoperability:
   Interoperability

(B) All files that have “Cloud Computing” in their text:
   cloud(s) computing

(C) Structured Query Language or SQL:
   S(structured) Query (language)

(D) All references to TCP/IP or Transmission Control Protocol Internet Protocol:
   Transmission(s) Control(s) Protocol(IP)

(E) All files that reference “Computer Network” book of “Andrew Stuart Tanenbaum”:
   Andrew Stuart Computer Networks

(F) All dates with YYYY/MM/DD format:
   19|20\(\d\d\d\d)/(0(1|2|3|4|5|6|7|8|9|1(0|1|2))ﾉ/
   (0(1|2|3|4|5|6|7|8|9|1(1|2))\d\d|0(0|1))

(G) URLs that include Computer Science (cs) or Electrical and Computer Engineering (ece) and finished by .edu:
   http://(www)\(\d\d\d\d)\d\d\d\d/(cs|ece)\.(www)\(\d\d\d\d).edu

(H) All IEEE conference papers after the year 2000:
   IEEE Conference (\s)+ (w|s)*

(I) Any XML scripts in the papers:
   <\s>(\s)* (xml|html) (\s)+ (\s)>

(J) Papers from any US city, state, and possibly ZIP code:
   \(\d\d\d\d)(\s)+ (\s)+ (\s)+ (w|s)+(\s)*d|d|d|d(-d|d|d|d)
RESeED Performance

- RESeED performs better than grep on unencrypted data
Conclusions

• We presented RESeED a tool to search regular expressions over encrypted data

• RESeED is
  ▪ Scalable \( (O(nw) \ll O(2^{nw})) \)
  ▪ imposes low storage overhead (6%)
  ▪ User-transparent and Cloud-agnostic
  ▪ Outperforms grep utility in several benchmarks
RESeED Commercialization on Fortivault (Fortinet Gateway)
**RESeED Demo:**


<table>
<thead>
<tr>
<th>File name</th>
<th>Date modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>uploads/rfc1152.txt</td>
<td>June-27-2014 18:00:00</td>
</tr>
<tr>
<td>uploads/rfc1155.txt</td>
<td>June-27-2014 18:00:07</td>
</tr>
<tr>
<td>uploads/rfc1156.txt</td>
<td>June-27-2014 18:00:10</td>
</tr>
<tr>
<td>uploads/rfc1157.txt</td>
<td>June-27-2014 18:00:12</td>
</tr>
<tr>
<td>uploads/rfc1158.txt</td>
<td>June-27-2014 18:00:14</td>
</tr>
<tr>
<td>uploads/rfc1160.txt</td>
<td>June-27-2014 18:00:18</td>
</tr>
<tr>
<td>uploads/rfc1161.txt</td>
<td>June-27-2014 18:00:21</td>
</tr>
<tr>
<td>uploads/rfc1162.txt</td>
<td>June-27-2014 18:00:23</td>
</tr>
<tr>
<td>uploads/rfc1163.txt</td>
<td>June-27-2014 18:00:25</td>
</tr>
<tr>
<td>uploads/rfc1164.txt</td>
<td>June-27-2014 18:00:29</td>
</tr>
<tr>
<td>uploads/rfc1165.txt</td>
<td>June-27-2014 18:00:31</td>
</tr>
<tr>
<td>uploads/rfc1166.txt</td>
<td>June-27-2014 18:00:34</td>
</tr>
</tbody>
</table>
Dropbox as a Dumb Storage
Proposal 1: Parallel RESeED

• How RESeED can handle big-data scale data sets?
• Proposal: parallelize data processing in RESeED
• Parallelization levels
  ▪ Column Store matching
  ▪ Order Store matching
• Hadoop/spark based processing for matching against Column store?
Parallelizing Bottlenecks

• Works based on several sub-column stores
• The number of Column stores constructed depends on the number of available machines
• When a file is uploaded, the appropriate sub-column store is determined based on the number of sub-column stores

\[ i = H(w) \mod N \]
Parallelizing Bottlenecks

Map

Sub-column store 1

Sub-column store 2

Sub-column store N-1

Sub-column store N

PE1

PE2

PE N-1

PE N

List of tokens that match any of the sub-NFAs

Reduce

Finds files that matches all sub-NFAs

List of files match sub-NFAs

...
Proposal 2: Applying RESeED for Health Care!

- Extend RESeED to search Genome sequences
- Proteins have starting and stopping codons

\[ \sum = C \cup \left( \omega_0 c_0 \cap c_1 \cap c_2 \cap \omega_1 \right) \]

- There is fuzziness in the coding of DNA sequences: N nucleotide can be \{A,C,G,T,U\}
Project Proposal 3: Semantic search over encrypted data

• Semantic search is required to explore increasingly stored data on the Cloud

• How can we do semantic search on encrypted data?

• Creating a semantic relationship library based on the Column store
Thank you for your time.

Any Question?
Appendix
Hash-width for arXiv
Timing for arXiv

![Timing for arXiv](image.png)
Scalability

![Scalability Graph]

- **Time (s)** is plotted on the y-axis.
- **Data-Set Size (GB)** is plotted on the x-axis.
- Different markers and colors represent different datasets (A to J).

The graph shows the relationship between data-set size and processing time, indicating scalability performance for various datasets.
Time to Update Column Store
We gain a better speedup for less fuzzy benchmarks
Parallelizing Bottlenecks

• Works based on several sub-column stores
  ▪ *The number of Column stores depends on the number of machines*

• When a file is uploaded, the appropriate sub-column store is determined

\[ i = H(w) \mod N \]