Multi-Resolution Similarity Hashing

Vassil Roussev
Golden G. Richard III
Lodovico Marziale
University of New Orleans
<vassil,golden,vico@cs.uno.edu>
Forensic Challenge #1: Scale

2007

750GB

750GB
Stream-oriented Tools: **MRS Hash**

- **Observation**
  - Cloning a target now is a (necessary) waste of time ...
  - ... almost nothing is known at the end

- **Rationale:**
  - Do something useful *while* cloning

- **Idea:**
  - Generate a similarity hash that can be put to immediate use
  - **Multi-Resolution Similarity Hash**
Requirements

- **Performance**
  - Work at line speed (or at least at block hash rates)

- **Scalability**
  - Work for any size target
  - Compare the tiny and the ENORMOUS

- **Efficiency**
  - Space efficiency (low overhead)
  - Processing efficiency (quick comparison)

- **Ease of use/standardization**
  - Must work out of the box
Basic Idea

- Repeatedly find a **context**
- Hash chunks in between contexts
- Compose hashes

\[
\text{Hash} = h(c_1) \cdot h(c_2) \cdot \ldots \cdot h(c_n)
\]
Rationale

- **Block hashes are fragile:**
  - Insert/delete at the beginning
  - Reordering

- **Context-based hashes are resilient:**
  - Insert/delete have only local effect
  
  ➔ We can discover ‘versions’
    - Modified file
    - Piece-to-whole correlation
    - Common pieces
Design/Implementation Options

- Each step offers many choices:
  - Context:
    » length, discovery algorithm
  - Hashing:
    » hash function(s), granularity
  - Composition
    » sequence vs. set
    » fixed vs. variable size
  - Comparison semantics
Example: ssdeep

- Context discovery:

  Adler32'

  t-bit selection

  $t$
ssdeep (2)

- **Chunk hashing**
  - FNV-derivative, LSB6
    - 6 bits of hash/chunk

- **Hash composition**
  - Base64 string concatenation
  - Fixed size hash

- **Comparison**
  - Edit distance
ssdeep (3)

- Some (constructive) critique:
  - Optimal context == ???
  - Adler32 & FNV are weak hashes
    » Can lead to skewed distributions of chunk sizes
    » Low-entropy data is a problematic
  - Hash concatenation
    » Do we really want to pay for sequence-based composition?
      – Requires 6-8 times more space than a set-based one
  - Fixed size
    » Causes repeated hash calculations (1.33 for html, 2.0 for doc/xls)
    » Catastrophic loss of accuracy for larger targets
  - Comparison
    » Edit distance—what does it tell us for binary data?
Building a Better Mousetrap

- **Context discovery**
  - Size of 7 bytes appears reasonable
  - **Hash:**
    - Adler32 is probably ok
    - We picked djb2 which works as well as md5
  - **Optimal** $t = ?$
    - Optimal $t \Leftrightarrow$ average chunk size $\sim 2^t$
    - NB: To compare $f_1$ & $f_2$, we *must* ensure $t_1 == t_2$!
      - ssdeep cannot do this for arbitrary files
    - Assume $t = 8$ (for now)
Hash Composition: Bloom Filters

\[ h_1(S_1) \quad h_2(S_1) \quad h_3(S_1) \quad \ldots \quad h_k(S_1) \]

\[ h_1(S_2) \quad h_2(S_2) \quad h_3(S_2) \quad \ldots \quad h_k(S_2) \]
Bloom Filters: False Positive Rates

<table>
<thead>
<tr>
<th>m/n</th>
<th>k</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>0.0138</td>
<td>0.0024</td>
<td>0.0009</td>
<td>0.0006</td>
<td>0.0005</td>
<td>0.0007</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>0.0177</td>
<td>0.0038</td>
<td>0.0018</td>
<td>0.0013</td>
<td>0.0013</td>
<td>0.0022</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>0.0236</td>
<td>0.0065</td>
<td>0.0037</td>
<td>0.0032</td>
<td>0.0041</td>
<td>0.0075</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.0329</td>
<td>0.0118</td>
<td>0.0085</td>
<td>0.0085</td>
<td>0.0136</td>
<td>0.0272</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.0490</td>
<td>0.0240</td>
<td>0.0216</td>
<td>0.0255</td>
<td>0.0484</td>
<td>0.0979</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.1549</td>
<td>0.1598</td>
<td>0.2201</td>
<td>0.3128</td>
<td>0.5423</td>
<td>0.7444</td>
</tr>
</tbody>
</table>
Comparing Bloom Filters

- Filters: \( f_1, f_2 \)
  - \( f_{12} = f_1 \cdot f_2 \) (bitwise AND)
  - Number of zeroes: \( Z_1, Z_2, Z_{12} \)
  - \( lz = \log\left[\frac{(Z_1 + Z_2 - Z_{12})}{(Z_1 Z_2)}\right] \)
  - \( \log(1/2^{11}) \leq lz \leq \log(1/Z_1) \)

\[ \Rightarrow 0 \leq Z\text{-score} \leq 1 \]
Bloom Filters as Composite Hashes

- md5 the chunks and place in a BF:

  ![Diagram of Bloom Filters with four hashes (h1, h2, h3, h4) and bit arrays]

- Example:
  - \( l = 11 \Rightarrow m = 2^l = 2048 \text{ bits} = 256 \text{ bytes} \)
  - \( k = 4, \ n = 256 \ (m/n = 8) \)
  - Recall that \( t = 8 \)
  - Expected coverage per BF:
    - \( n \times 2^t = 256 \times 256 = 64\text{KB} \)
Comparing Similarity Hashes

\[ f_1, f_2, \ldots, f_m, g_1, g_2, \ldots, g_n \]
Resolution & Scalability

- Z-score has quadratic complexity: $O(nm)$
- For $t = 8$, exp. coverage: $256 \times 2^8 = 64$ KB:
  - 64KB vs. 64KB $\Rightarrow$ $\sim$ 1 comparison
  - 64KB vs. 64MB $\Rightarrow$ $\sim$ 1,000 comps
  - 64MB vs. 64MB $\Rightarrow$ $\sim$ 1 mln comps
  - ...
  - 64 GB vs. 64 GB $\Rightarrow$ ???
- For $t = 12 \Rightarrow$ exp. coverage: $256 \times 2^{12} = 1$ MB
  - 64 MB vs. 64 MB $\Rightarrow$ $\sim$ 3,600 comps
  - 64 GB vs. 64 GB $\Rightarrow$ $\sim$ 4.300 bln comps
- For $t = 16 \Rightarrow$ exp. coverage: $256 \times 2^{16} = 16$ MB
  - 64 GB vs. 64 GB $\Rightarrow$ $\sim$ 4.3 mln comps
  ...

...
Multi-Resolution Similarity Hash

- Q: Optimal $t$?
- A: No single value will work
- Q: Solution?
- A: Take multiple resolutions
- Q: Which ones?
- A: Pick a “reasonable” **standard** set of numbers:
  - $t = 8, 12, 16, 20, 24, 28, 32$
  - Note: for $t = 32$, coverage is $2^{40}/BF = 1$TB (!)
  - Note: not all hashes will have all resolutions
    » We set a minimum of 16 BF elements
MRS Hash: Raw Performance

- Hash generation: > 20MB/s
  - Pentium D 2.8GHz
  - Dominant cost
  - Comparable to 512-block MD5
  - Alpha version:
    - single thread
    - almost no optimizations
    - it’s “embarrassingly” parallelizable

- Storage requirements:
  - < 0.5% of target, i.e. 500GB → 2.5GB
MRS Hash: What does it mean?

- **Observation:**
  - Our comparison is purely syntactic, and
  - Can be applied to *any* data.
  - We cannot predict what it means!

- **Q:** How do we know it’s useful?

- **A:** Empirical study
  - Shows how the Z-score should be interpreted
  - Demonstrates possible uses
MRS Hash: Empirical Study

- Corpus: downloaded internet files (Yahoo!)
  - File types: doc, xls, pdf, jpg, (html)
  - 10 files per (generic) topic per type
  - Data cleaned up manually

- Scenarios (for each type)
  - ‘All-pairs’: compare all file pairs
  - ‘Half-directory’:
    » Place (random) half of file in an uncompressed zip
    » Compare all files to the zip file

Q: Can Z-score split true positive & true negatives?
Empirical Study: \textit{doc} (all-pairs)

- 355 files, 64kB - 10MB, 298MB total
- All-pairs:
  - 62,835 pairs (ps)
  - 57 ps (<0.1%) with z-score > 0.1
  - 18 ps with z-score > 0.2:
    - 16 TP (true positives)
    - 2 FP (false positives)
  - 29 ps b/w 0.1 & 0.2:
    - 1 TP
    - 28 TN

\textbf{\textbf{Threshold of 0.2 yields: 2 FP, 1 FN!}}

- File versions found: XBRL 2.1 (92/99p), manual (53/54)
Empirical Study: doc (half-dir)

- Note: all known similar files have been removed
Empirical Study: xls (all-pairs)

- 415 files, 64kB - 7MB, 257MB total
- All-pairs:
  - 85,905 pairs (ps)
  - 26 ps (<<0.1%) with z-score > 0.1
  - Threshold of 0.2 yields: 1 FP, 1 FN
  - Found: different drafts of an environment form
Empirical Study: xls (half-dir)
Empirical Study: jpeg (all-pairs)

- 737 files, 64kB - 5MB, 121MB total
- All-pairs:
  - 273,370 ps
  - 46 ps (<0.01%) with z-score > 0.1
    - 4 TP (0.214, 0.166, 0.136, 0.121) among top 5
    - Example?
Empirical Study: jpeg (half-dir)
Empirical Study: pdf

- Only 59 files:
  - All-pairs: no FP/FN
  - Half-dir: TN < 0.03, TP > 0.20

- Clustering
  - Added files w/ common format:
  - Cluster #1: 9 book chapters
  - Cluster #2: 5 DFRWS’06 papers

- Results:
  - C1: 0.88-0.90, 0.66-0.69, 0.55
  - C2: 0.12-0.19
  - All others (2,003 out of 2,046) : < 0.09
Conclusions (Req. Review)

- **Performance**
  - Work at line speed (or at least at block hash rates)

- **Scalability**
  - Work for any size target
  - Compare the tiny and the ENORMOUS

- **Efficiency**
  - Space efficiency (low overhead)
  - Processing efficiency (quick comparison)

- **Ease of use/standardization**
  - Must work out of the box
Conclusions (2)

- MRSIH can also help you conduct:
  - Efficient searches for files/fragments;
  - Privacy preserving inquiries;
  - Live forensics (taking system signature);
  - Object version discovery;
  - Large-scale target correlation;
  - ...


Future Work

- Define serialized format
- Optimize
  - Line speed should be achievable
- Parallelize
  - GPU processing can massively speedup comparisons
- Test for scale
  - Drive-scale testing is necessary
  - Sub-file testing
Thank You!

Questions?