EFFICIENT PARALLEL SET-SIMILARITY JOINS USING MAPREDUCE

Rares Vernica, Michael J. Carey, Chen Li

Seminar Large-Scale Duplicate Detection Paper Presentation
Maria Graber, Lukas Schulze
Duplicates on freedb

• Find Similar Track Titles in freedb
• Join information with similar track title

<table>
<thead>
<tr>
<th>Song Title</th>
<th>Length</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Message In The Bottle</td>
<td>3:45</td>
<td>The Police</td>
</tr>
<tr>
<td>Fading Like A Flower</td>
<td>2:56</td>
<td>Roxette</td>
</tr>
<tr>
<td>The Way</td>
<td>4:12</td>
<td>Fastball</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Song Title</th>
<th>Length</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Message In a Bottle</td>
<td>3:45</td>
<td>The Police</td>
</tr>
<tr>
<td>Fading Like A Flower</td>
<td>2:56</td>
<td>Roxette</td>
</tr>
<tr>
<td>The Way</td>
<td>4:12</td>
<td>Fastball</td>
</tr>
</tbody>
</table>
Set-Similarity Join

- Join on set attribute a, e.g. array or *tokenized* string
  - “The Message In The Bottle“ → [The, Message, In, Bottle]

- Given:
  - Records R and S
  - Set-similarity function sim
  - Threshold t

  Jaccard similarity function:
  \[ \text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|} \]
  \( x = \text{[The, Message, In, Bottle]} \)
  \( y = \text{[Message, In, A, Bottle]} \)
  \( \text{jaccard}(x,y) = \frac{3}{5} = 0.6 \)

→ Join all pairs with \( \text{sim}(R.a, S.a) \geq t \)
Efficient Set-Similarity Join

- Idea: Use subset instead of whole set to reduce number of similarity checks

→ Prefix Filtering
- „[…] similar strings need to share at least one common token in their prefixes“
  - Necessary criterion

- [Message, In, A, Bottle], Prefix: [Message, A, Bottle]
- [The Message, In, Bottle], Prefix: [Message, Bottle, In]
Parallel Join using MapReduce

• Steps:
  1. Token Ordering
  2. RID-Pair Generation
  3. Record Join

• Each step uses (multiple) MapReduce jobs
Step 1: Token Ordering

- Create list of all tokens in join attribute values

<table>
<thead>
<tr>
<th>Song Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Message In The Bottle</td>
</tr>
<tr>
<td>Message In A Bottle</td>
</tr>
<tr>
<td>Fading Like A Flower</td>
</tr>
<tr>
<td>The Way</td>
</tr>
</tbody>
</table>

→ [The, Message, In, Bottle, A, Fading, Like, Flower, Way]

- Used to define prefix
- Sort ascendingly by frequency (low → high)
  → [Fading, Like, Flower, Way, Message, Bottle, A, In, The]
Step 1: Token Ordering

- Input: join attribute values

- Word Count
  - Map: Each Word has count 1
  - Reduce: Add counts for each word

- Sort
  - Map: Swap key (word) and value (count)
  - Reduce: One reducer sorts list ascendingly

- Output: [Fading, Like, Flower, Way, Message, Bottle, A, In, The]
Step 2: RID-Pair Generation

- Find record IDs with similar join attribute
  - Extract prefix
  - Compute similarity of two join attribute values with one common token in prefix (Prefix Filtering)
  - Output pairs of Record IDs
Step 2: RID-Pair Generation

Map Phase: Extract prefix

- Determine prefix length of tokenized string s
  - Used threshold: \( t = 0.5 \)
  - Used algorithm: Jaccard
  - Prefix Length: \( |s| - \lceil |s| \times t \rceil + 1 \)

  - \( 4 - \lceil 4 \times 0.5 \rceil + 1 = 3 \)
  → Prefix: [Message, Bottle, A]

- “The Message In The Bottle“ → [The, Message, In, Bottle]
  - \( 4 - \lceil 4 \times 0.5 \rceil + 1 = 3 \)
  → Prefix: [Message, Bottle, In]

- Global token ordering: [Fading, Like, Flower, Way, Message, Bottle, A, In, The]
Step 2: RID-Pair Generation

- Intermediate output for each prefix token: record ID + set
  - “Message In A Bottle“:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>2, Message, In, A, Bottle</td>
</tr>
<tr>
<td>Bottle</td>
<td>2, Message, In, A, Bottle</td>
</tr>
<tr>
<td>A</td>
<td>2, Message, In, A, Bottle</td>
</tr>
</tbody>
</table>

- “The Message In The Bottle“:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>1, The, Message, In, Bottle</td>
</tr>
<tr>
<td>Bottle</td>
<td>1, The, Message, In, Bottle</td>
</tr>
<tr>
<td>In</td>
<td>1, The, Message, In, Bottle</td>
</tr>
</tbody>
</table>
Step 2: RID-Pair Generation

- **Reduce Phase**: Find pairs
  - Grouped by prefix token
  → Prefix Filtering condition for similar strings
  - Optional: Apply more filters (PPJoin algorithm)
  - Calculate similarity (Cartesian product)
    - similarity ≥ threshold → output pair of record IDs

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>2, Message, In, A, Bottle</td>
</tr>
<tr>
<td>Message</td>
<td>1, The, Message, In, Bottle</td>
</tr>
</tbody>
</table>

- Jaccard: $3/5 = 0.6$
- Threshold: 0.5
- Output: (1,2) ✔
Step 3: Record Join

- Join data of computed record ID pairs
- **Map Phase**: Join one record with record ID pair
  - Load list of record ID pairs on each map node from HDFS

<table>
<thead>
<tr>
<th>RID</th>
<th>Song Title</th>
<th>Length</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Message In The Bottle</td>
<td>3:45</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Message In A Bottle</td>
<td>3:45</td>
<td>The Police</td>
</tr>
</tbody>
</table>

- Output:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>1, The, Message, In, Bottle, 3:45,</td>
</tr>
<tr>
<td>1,2</td>
<td>2, Message, In, A, Bottle, 3:45, The Police</td>
</tr>
</tbody>
</table>
Step 3: Record Join

- Join data of computed record ID pairs
- **Reduce Phase**: Join both records

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>1, The, Message, In, Bottle, 3:45,</td>
</tr>
<tr>
<td>1,2</td>
<td>2, Message, In, A, Bottle, 3:45, The Police</td>
</tr>
</tbody>
</table>

- **Output:**

<table>
<thead>
<tr>
<th>Song Title</th>
<th>Length</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Message In The Bottle</td>
<td>3:45</td>
<td>The Police</td>
</tr>
</tbody>
</table>
Summary

- Paper authors benchmarked variations of the algorithm
  - Similar runtime
  → We can use easiest implementation!

- Steps:
  - Global Token Ordering
  - Calculate similarity of set pairs with common prefix token
  - (Join data)
Our Use Case

• Data Preparation
  • Lower case
  • Stop-word-filtering (articles, special characters,…)

• Tokenize strings
  • Words
  • Resort characters in tokens (e.g. bottle = belott)
  • N-grams (e.g. bigrams: Bottle = Bo, ot, tt, …)

• Global token ordering
  • Privilege longer tokens

• Similarity function
  • Choice of threshold
  • Jaccard for track title? → better for large sets, e.g. whole track list
Sources

PPJoin: Length Filter

- Minimal number of common token for given set
  - $S_1 = \{A, B, C, D, E\}$
  - Threshold $t = 0.8$
  - Number of common token $\geq 4$
  - Discard set with length $< 4$
PPJoin: Positional Filter

- Order sets according to ascending global token frequency
- Position of common token can discard pair
- \[ \text{O}(S1,S2) \geq \frac{t}{t+1} \cdot (|x| + |y|) \]
  - \( S1 = \{A,B,C,D,E\} \)
  - \( S2 = \{B,C,D,E,F\} \)
  - Threshold \( t = 0.8 \)
  \[ \rightarrow \text{Number of common token } \text{O}(S1,S2) \geq 5 \]

\[ \rightarrow \text{Number of common token in subset until specific token } + \text{ minimal } \]
\[ \text{“unseen“ token } \geq \text{ minimal } \text{O}(S,S2) \]

\[ \rightarrow 1 + \min(3,4) = 4 \geq 5 \]
\[ \rightarrow \text{Discard pair } S1, S2 \]
PPJoin: Suffix Filter

- Hamming Distance
  - Difference between 2 strings
    - clear vs. clean $\rightarrow$ Hamming Distance = 1

- Prerequisites
  - 2 records $x$ and $y$
  - Token length: $|y| \leq |x|$

- Suffix of an entry are all tokens which are not in the prefix
PPJoin: Suffix Filter

• Upper bound of HD derived from minimum length of prefixes:
  \[ H_{\text{max}}(x_s, y_s) = |x| + |y| - 2 \times \left[ \frac{t}{t+1} \times (|x| + |y|) \right] - (\lceil t \times |x| \rceil - \lceil t \times |y| \rceil) \]

• Lower bound of HD can be estimated as the difference of length in partitions:
  \[ H_{\text{low}}(x_s, y_s) = \text{abs}(|x_l| - |y_l|) + \text{abs}(|x_r| - |y_r|) = \text{abs}(|x_{ll}| - |y_{ll}|) + \text{abs}(|x_{lr}| - |y_{lr}|) + \text{abs}(|x_r| - |y_r|) \]
PPJoin: Suffix Filter

1. Choose a token w in y (e.g. the middle one)
   - global token ordering: (F, C, B, G, E, A, H, D)
   - \( x = (A, C, G, B, E, D) \)
   - \( y = (C, H, E, A, F, D) \)
   - \( w = E \)

2. Create 2 partitions
   - Partition #1: all tokens, which are before w in global token ordering
     - \( x_l = (C, B, G) \)
     - \( y_l = (F, C) \)
   - Partition #2: all tokens, which are after w in gto and w itself
     - \( x_r = (E, A, D) \)
     - \( y_r = (E, A, H, D) \)

3. Calculate lower bound of Hamming Distance
   - \( H_{\text{low}} = \text{abs}(|x_l| - |y_l|) + \text{abs}(|x_r| - |y_r|) = \text{abs}(3 - 3) + \text{abs}(2 - 4) = 2 \)

4. Compare with \( H_{\text{max}} \) and create 2 new partitions out of \( x_l \)
   - if \( H_{\text{low}} \leq H_{\text{max}} \)