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

Song Title	Length
The Message In The Bottle	3:45
Fading Like A Flower	2:56
The Way	4:12

Song Title	Artist
Message In A Bottle	The Police
Fading Like A Flower	Roxette
The Way	Fastball

Song Title	Length	Artist
The Message In The Bottle	3:45	The Police
Fading Like A Flower	2:56	Roxette
The Way	4:12	Fastball

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 = [The, Message, In, Bottle]
y = [Message, In, A, Bottle]
jaccard(x,y) = 3/5 = 0.6

→ Join all pairs with sim(R.a, S.a) ≥ 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

Song Title
The Message In The Bottle
Message In A Bottle
Fading Like A Flower
The Way

→ [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

Song Title
The Message In The Bottle
Message In A Bottle
Fading Like A Flower
The Way

- 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| - ⌈ |s| * t ⌉ + 1

- "Message In A Bottle" → [Message, In, A, Bottle]
  - 4 - ⌈ 4 * 0.5 ⌉ + 1 = 3

  → **Prefix**: [Message, Bottle, A]

- "The Message In The Bottle" → [The, Message, In, Bottle]
  - 4 - ⌈ 4 * 0.5 ⌉ + 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“:

Key	Value
Message	2, Message, In, A, Bottle
Bottle	2, Message, In, A, Bottle
A	2, Message, In, A, Bottle

  - “The Message In The Bottle“:

Key	Value
Message	1, The, Message, In, Bottle
Bottle	1, The, Message, In, Bottle
In	1, The, Message, In, Bottle

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)
    - similiarity ≥ threshold → output pair of record IDs

Key	Value
Message	2, Message, In, A, Bottle
Message	1, The, Message, In, Bottle

    - 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

RID	Song Title	Length	Artist
1	The Message In The Bottle	3:45	
2	Message In A Bottle	3:45	The Police

  - Output:

Key	Value
1,2	1, The, Message, In, Bottle, 3:45,
1,2	2, Message, In, A, Bottle, 3:45, The Police

Step 3: Record Join

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

Key	Value
1,2	1, The, Message, In, Bottle, 3:45,
1,2	2, Message, In, A, Bottle, 3:45, The Police

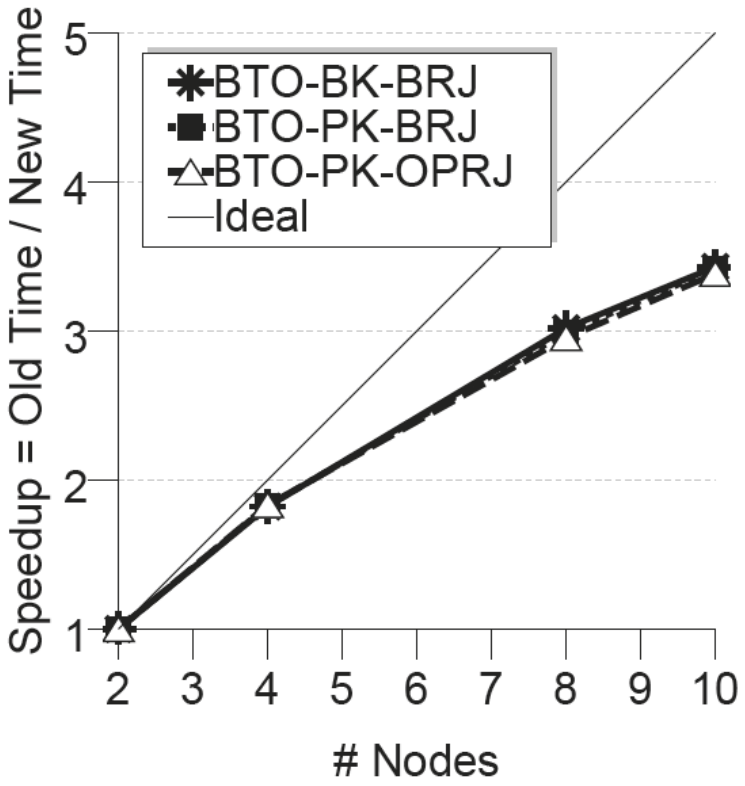
  - Output:

Song Title	Length	Artist
The Message In The Bottle	3:45	The Police

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

- Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using MapReduce. In *Proceedings of the International Conference on Management of data*(SIGMOD), 2010.
- Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.* 36, 3, Article 15 (August 2011)

PPJoin: Length Filter

- Minimal number of common token for given set
  - S1 = {A,B,C,D,E}
  - Threshold t = 0.8

  → Number of common token ≥ 4

  → Discard set with length < 4

PPJoin: Positional Filter

- Order sets according to ascending global token frequency
- Position of common token can discard pair
- O(S1,S2) ≥ $\frac{t}{t+1} * (|x| + |y|)$
  - S1 = {A,B,C,D,E}
  - S2 = {B,C,D,E,F}
  - Threshold t = 0.8

  → Number of common token O(S1,S2) ≥ 5

→ Number of common token in subset until specific token + minimal „unseen“ token ≥ minimal O(S,S2)

→ 1 + min(3,4) = 4 ≥ 5 ↯

→ Discard pair S1, S2

PPJoin: Suffix Filter

- Hamming Distance
  - Difference between 2 strings
    - clear vs. clean → 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:
  - $H_{max}(x_s,y_s) = |x| + |y| - 2 * \lceil \frac{t}{t+1} * (|x| + |y|) \rceil - (\lceil t * |x| \rceil - \lceil t * |y| \rceil)$
- Lower bound of HD can be estimated as the difference of length in partitions:
  - $H_{low}(x_s, y_s) = abs(|x_l| - |y_l|) + abs(|x_r| - |y_r|)$
    $= abs(|x_{ll}| - |y_{ll}|) + abs(|x_{lr}| - |y_{lr}|) + abs(|x_r| - |y_r|)$

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
  - $H_{low} = abs(|x_l| - |y_l|) + abs(|x_r| - |y_r|) = abs(3 - 3) + abs(2 - 4) = 2$
- 4. Compare with H_{max} and create 2 new partitions out of x_l if $H_{low} \leq H_{max}$