Efficient Parallel Set-Similarity Joins Using MapReduce

Rares Vernica

Department of Computer Science
University of California, Irvine

SIGMOD 2010
Joint work with Michael J. Carey and Chen Li (UC Irvine)
Outline

1. Motivation
2. Problem Statement
   - Inverted List Index
   - Set-Similarity Filters
4. Parallel Algorithms
5. Experimental Evaluation
### Example 1: Data Cleaning/Master-Data-Management

#### Customer data from two departments

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sales</th>
<th>ID</th>
<th>Name</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>S10</td>
<td>John W Smith</td>
<td>...</td>
<td>R20</td>
<td>Smith John</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Master customer data across two departments

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C30</td>
<td>John W Smith</td>
</tr>
</tbody>
</table>
### Example 1: Data Cleaning/Master-Data-Management

Customer data from two departments

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sales</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>S10</td>
<td>John W Smith</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>R20</td>
<td>Smith John</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Master customer data across two departments

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C30</td>
<td>John W Smith</td>
</tr>
</tbody>
</table>
### Example 1: Data Cleaning/Master-Data-Management

**Customer data from two departments**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>S10</td>
<td>John W Smith</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R20</td>
<td>Smith John</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Master customer data across two departments**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>C30</td>
<td>John W Smith</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rares Vernica  (UC Irvine)
Example 1: Data Cleaning/Master-Data-Management

String → Set
- S10: "John W Smith" → \{John, Smith, W\}
- R20: "Smith John" → \{John, Smith\}

Set-Similarity Metric
- Jaccard similarity/Tanimoto coefficient: jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|}
- jaccard(S10, R20) = \frac{2}{3}

Set-Similarity Join
Set-Similarity Join “Sales” and “Returns” on “Name”
Example 1: Data Cleaning/Master-Data-Management

**String → Set**

- **S10:** “John W Smith” → \{John, Smith, W\}
- **R20:** “Smith John” → \{John, Smith\}

**Set-Similarity Metric**

- **Jaccard similarity/Tanimoto coefficient:** \( jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
- \( jaccard(S10, R20) = \frac{2}{3} \)

**Set-Similarity Join**

Set-Similarity Join “Sales” and “Returns” on “Name”
Example 1: Data Cleaning/Master-Data-Management

**String → Set**
- S10: “John W Smith” → {John, Smith, W}
- R20: “Smith John” → {John, Smith}

**Set-Similarity Metric**
- Jaccard similarity/Tanimoto coefficient: \( jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
- \( jaccard(S10, R20) = \frac{2}{3} \)

**Set-Similarity Join**
Set-Similarity Join “Sales” and “Returns” on “Name”
Example 1: Data Cleaning/Master-Data-Management

String → Set

- S10: “John W Smith” → {John, Smith, W}
- R20: “Smith John” → {John, Smith}

Set-Similarity Metric

- Jaccard similarity/Tanimoto coefficient: \( \text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
- \( \text{jaccard}(S10, R20) = \frac{2}{3} \)

Set-Similarity Join

Set-Similarity Join “Sales” and “Returns” on “Name”
**Example 1: Data Cleaning/Master-Data-Management**

**String → Set**
- S10: “John W Smith” → \{John, Smith, W\}
- R20: “Smith John” → \{John, Smith\}

**Set-Similarity Metric**
- Jaccard similarity/Tanimoto coefficient: \( jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
- \( jaccard(S10, R20) = \frac{2}{3} \)

**Set-Similarity Join**
Set-Similarity Join “Sales” and “Returns” on “Name”
Example 1: Data Cleaning/Master-Data-Management

String → Set
- \textbf{S10:} “John W Smith” → \{John, Smith, W\}
- \textbf{R20:} “Smith John” → \{John, Smith\}

Set-Similarity Metric
- Jaccard similarity/Tanimoto coefficient: \( \text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
- \( \text{jaccard}(S10, R20) = \frac{2}{3} \)

Set-Similarity Join
Set-Similarity Join “Sales” and “Returns” on “Name”
Example 1: Data Cleaning/Master-Data-Management

String → Set
- S10: “John W Smith” → {John, Smith, W}
- R20: “Smith John” → {John, Smith}

Set-Similarity Metric
- Jaccard similarity/Tanimoto coefficient: \( jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
- \( jaccard(S10, R20) = \frac{2}{3} \)

Set-Similarity Join
Set-Similarity Join “Sales” and “Returns” on “Name”
Example 2: Social Networking Recommendations

### Current user’s profile

<table>
<thead>
<tr>
<th>Name</th>
<th>Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>UCI</td>
</tr>
<tr>
<td>Emp.</td>
<td>IBM</td>
</tr>
<tr>
<td>Hobby</td>
<td>Running</td>
</tr>
</tbody>
</table>

### Related user’s profiles

<table>
<thead>
<tr>
<th>Name</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>Harvard</td>
</tr>
<tr>
<td>Emp.</td>
<td>IBM</td>
</tr>
<tr>
<td>Hobby</td>
<td>Running</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>UCI</td>
</tr>
<tr>
<td>Emp.</td>
<td>Yahoo</td>
</tr>
<tr>
<td>Hobby</td>
<td>Running</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Mike</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>MIT</td>
</tr>
<tr>
<td>Emp.</td>
<td>IBM</td>
</tr>
<tr>
<td>Member</td>
<td>ACM</td>
</tr>
</tbody>
</table>
Example 2: Social Networking Recommendations

Current user’s profile

<table>
<thead>
<tr>
<th>Name:</th>
<th>Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>School:</td>
<td>UCI</td>
</tr>
<tr>
<td>Emp.:</td>
<td>IBM</td>
</tr>
<tr>
<td>Hobby:</td>
<td>Running</td>
</tr>
</tbody>
</table>

Related user’s profiles

<table>
<thead>
<tr>
<th>Name:</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>School:</td>
<td>Harvard</td>
</tr>
<tr>
<td>Emp.:</td>
<td>IBM</td>
</tr>
<tr>
<td>Hobby:</td>
<td>Running</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>School:</td>
<td>UCI</td>
</tr>
<tr>
<td>Emp.:</td>
<td>Yahoo</td>
</tr>
<tr>
<td>Hobby:</td>
<td>Running</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Mike</th>
</tr>
</thead>
<tbody>
<tr>
<td>School:</td>
<td>MIT</td>
</tr>
<tr>
<td>Emp.:</td>
<td>IBM</td>
</tr>
<tr>
<td>Member:</td>
<td>ACM</td>
</tr>
</tbody>
</table>
Example 2: Social Networking Recommendations

Profile → Set

- Alice: \{\text{Emp.}:\text{IBM}, \text{Hobby}:\text{Running}, \text{School}:\text{UCI}\}
- Bob: \{\text{Emp.}:\text{IBM}, \text{Hobby}:\text{Running}, \text{School}:\text{Harvard}\}
- John: \{\text{Emp.}:\text{Yahoo}, \text{Hobby}:\text{Running}, \text{School}:\text{UCI}\}
- Mike: \{\text{Emp.}:\text{IBM}, \text{Member}:\text{ACM}, \text{School}:\text{MIT}\}

Set-Similarity Metric (\text{jaccard})

- \text{jaccard}(Alice, Bob) = \frac{2}{4}
- \text{jaccard}(Alice, John) = \frac{2}{4}
- \text{jaccard}(Alice, Mike) = \frac{1}{5}

Set-Similarity Self-Join

Set-Similarity Self-Join users on profile
Example 2: Social Networking Recommendations

Profile $\rightarrow$ Set

- Alice: $\{\text{Emp.:IBM, Hobby:Running, School:UCI}\}$
- Bob: $\{\text{Emp.:IBM, Hobby:Running, School:Harvard}\}$
- John: $\{\text{Emp.:Yahoo, Hobby:Running, School:UCI}\}$
- Mike: $\{\text{Emp.:IBM, Member:ACM, School:MIT}\}$

Set-Similarity Metric ($jaccard$)

- $jaccard(\text{Alice, Bob}) = \frac{2}{4}$
- $jaccard(\text{Alice, John}) = \frac{2}{4}$
- $jaccard(\text{Alice, Mike}) = \frac{1}{5}$

Set-Similarity Self-Join

Set-Similarity Self-Join users on profile
Example 2: Social Networking Recommendations

Profile → Set

- Alice: \{Emp.:IBM, Hobby:Running, School:UCI\}
- Bob: \{Emp.:IBM, Hobby:Running, School:Harvard\}
- John: \{Emp.:Yahoo, Hobby:Running, School:UCI\}
- Mike: \{Emp.:IBM, Member:ACM, School:MIT\}

Set-Similarity Metric (\textit{jaccard})

- \textit{jaccard}(Alice, Bob) = \frac{2}{4}
- \textit{jaccard}(Alice, John) = \frac{2}{4}
- \textit{jaccard}(Alice, Mike) = \frac{1}{5}

Set-Similarity Self-Join

Set-Similarity Self-Join users on profile
Example 2: Social Networking Recommendations

Profile $\rightarrow$ Set

- Alice: \{Emp.:IBM, Hobby:Running, School:UCI\}
- Bob: \{Emp.:IBM, Hobby:Running, School:Harvard\}
- John: \{Emp.:Yahoo, Hobby:Running, School:UCI\}
- Mike: \{Emp.:IBM, Member:ACM, School:MIT\}

Set-Similarity Metric (\textit{jaccard})

- $\text{jaccard}(Alice, Bob) = \frac{2}{4}$
- $\text{jaccard}(Alice, John) = \frac{2}{4}$
- $\text{jaccard}(Alice, Mike) = \frac{1}{5}$

Set-Similarity Self-Join

Set-Similarity Self-Join users on profile
Example 2: Social Networking Recommendations

<table>
<thead>
<tr>
<th>Profile → Set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alice:</strong> {Emp.: IBM, Hobby: Running, School: UCI}</td>
</tr>
<tr>
<td><strong>Bob:</strong> {Emp.: IBM, Hobby: Running, School: Harvard}</td>
</tr>
<tr>
<td><strong>John:</strong> {Emp.: Yahoo, Hobby: Running, School: UCI}</td>
</tr>
<tr>
<td><strong>Mike:</strong> {Emp.: IBM, Member: ACM, School: MIT}</td>
</tr>
</tbody>
</table>

Set-Similarity Metric (*jaccard*)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$jaccard(Alice, Bob)$</td>
<td>$\frac{2}{4}$</td>
</tr>
<tr>
<td>$jaccard(Alice, John)$</td>
<td>$\frac{2}{4}$</td>
</tr>
<tr>
<td>$jaccard(Alice, Mike)$</td>
<td>$\frac{1}{5}$</td>
</tr>
</tbody>
</table>

Set-Similarity Self-Join

Set-Similarity Self-Join users on profile
Example 2: Social Networking Recommendations

Profile → Set

- Alice: \{Emp.:IBM, Hobby:Running, School:UCI\}
- Bob: \{Emp.:IBM, Hobby:Running, School:Harvard\}
- John: \{Emp.:Yahoo, Hobby:Running, School:UCI\}
- Mike: \{Emp.:IBM, Member:ACM, School:MIT\}

Set-Similarity Metric (jaccard)

- \( jaccard(Alice, Bob) = \frac{2}{4} \)
- \( jaccard(Alice, John) = \frac{2}{4} \)
- \( jaccard(Alice, Mike) = \frac{1}{5} \)

Set-Similarity Self-Join

Set-Similarity Self-Join users on profile
Problem Statement: Set-Similarity Join

<table>
<thead>
<tr>
<th>RID</th>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="r1" alt="Shapes" /></td>
<td><img src="r1" alt="Shapes" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="r2" alt="Shapes" /></td>
<td><img src="r2" alt="Shapes" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="r3" alt="Shapes" /></td>
<td><img src="r3" alt="Shapes" /></td>
</tr>
<tr>
<td>...</td>
<td><img src="r..." alt="Shapes" /></td>
<td><img src="r..." alt="Shapes" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RID</th>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td><img src="s10" alt="Shapes" /></td>
<td><img src="s10" alt="Shapes" /></td>
</tr>
<tr>
<td>20</td>
<td><img src="s20" alt="Shapes" /></td>
<td><img src="s20" alt="Shapes" /></td>
</tr>
<tr>
<td>30</td>
<td><img src="s30" alt="Shapes" /></td>
<td><img src="s30" alt="Shapes" /></td>
</tr>
<tr>
<td>...</td>
<td><img src="s..." alt="Shapes" /></td>
<td><img src="s..." alt="Shapes" /></td>
</tr>
</tbody>
</table>
Problem Statement: Set-Similarity Join

**R**

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>2</td>
<td>△</td>
<td>□</td>
</tr>
<tr>
<td>3</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**S**

<table>
<thead>
<tr>
<th>RID</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>20</td>
<td>△</td>
<td>□</td>
</tr>
<tr>
<td>30</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**R.a~S.c**

Sim = Jaccard

τ = .5

<table>
<thead>
<tr>
<th>RID_R</th>
<th>a</th>
<th>b</th>
<th>RID_S</th>
<th>c</th>
<th>d</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem Statement: Set-Similarity Join

R.RID | R.a | R.b | S.RID | S.c | S.d | Sim
----|----|----|----|----|----|----
1   | a  | b  | 10  | c  | d  |  
2   |    |    | 20  |    |    |  
3   |    |    | 30  |    |    |  
... |    |    | ... |    |    |  

R.RID R.a ~ S.c
Sim = Jaccard
τ = .5

Rares Vernica (UC Irvine)
Problem Statement: Set-Similarity Join

R = \{a, b\}

S = \{c, d\}

Sim = \text{Jaccard}

\tau = 0.5
Problem Statement: Set-Similarity Join

Input
- Two files of records e.g., $R(RID, a, b)$ and $S(RID, c, d)$
- A join column on each file e.g., $R.a$ and $S.c$
- A similarity function, $sim$ e.g., Jaccard
- A similarity threshold, $\tau$

Output
All pairs of records from $R$ and $S$ where $sim(R.a, S.c) \geq \tau$
Outline

1. Motivation
2. Problem Statement
   - Inverted List Index
   - Set-Similarity Filters
4. Parallel Algorithms
5. Experimental Evaluation

Rares Vernica (UC Irvine)
Single Machine Set-Similarity Join

1. Nested loops
2. Inverted list index [Sarawagi and Kirpal, 2004]
   - Indexing phase
   - Candidate generation phase
   - Verification phase
Single Machine Set-Similarity Join

1. Nested loops
2. Inverted list index [Sarawagi and Kirpal, 2004]
   - Indexing phase
   - Candidate generation phase
   - Verification phase

- 10
- 20
- 30
- 40
- 50
Single Machine Set-Similarity Join

1. Nested loops
2. Inverted list index [Sarawagi and Kirpal, 2004]
   1. Indexing phase
   2. Candidate generation phase
   3. Verification phase

- **10**
  - 10
  - 10
  - 10
  - 10

- **20**
  - 20
  - 20
  - 20
  - 20

- **30**
  - 30
  - 30
  - 30
  - 30

- **40**
  - 40
  - 40
  - 40
  - 40

- **50**
  - 50
  - 50
  - 50
  - 50
Single Machine Set-Similarity Join

1. Nested loops
2. Inverted list index [Sarawagi and Kirpal, 2004]
   1. Indexing phase
   2. Candidate generation phase
   3. Verification phase

```
    10  
   □ △ □
```

```
   10 10 10
  30 40 30
  40 50 40
  50
```
Single Machine Set-Similarity Join

1. Nested loops
2. Inverted list index [Sarawagi and Kirpal, 2004]
   1. Indexing phase
   2. Candidate generation phase
   3. Verification phase

Rares Vernica (UC Irvine)
Set-Similarity Filtering

Prefix Filtering [Chaudhuri et al., 2006]

- Pigeonhole principle
- Global order for set elements:

  ![Shapes](image)

  Popularity

- Sort each record’s token
Prefix Filtering [Chaudhuri et al., 2006]

- Pigeonhole principle
- Global order for set elements:

  ![Diagram](image)

  Popularity

- Sort each record’s token
- E.g., sim is overlap size, \( \tau = 4 \)

```
10
```

```
20
```
Prefix Filtering [Chaudhuri et al., 2006]

- Pigeonhole principle
- Global order for set elements:
  - Sort each record’s token
  - E.g., $sim$ is overlap size, $\tau = 4$
  - Prefix length is 2

10

20
Set-Similarity Filtering

Prefix Filtering [Chaudhuri et al., 2006]

- Pigeonhole principle
- Global order for set elements:
  
  E.g., \( \text{sim} \) is overlap size, \( \tau = 4 \)
  
  Prefix length is 2

Sort each record’s token

10

20
Set-Similarity Filtering

Prefix Filtering [Chaudhuri et al., 2006]

- Pigeonhole principle
- Global order for set elements:
  
  ![Diagram of set elements]

  - Sort each record’s token
  - E.g., \( \text{sim} \) is overlap size, \( \tau = 4 \)
  - Prefix length is 2

  - 10
    
    ![Diagram of 10 elements]

  - 20
    
    ![Diagram of 20 elements]
Set-Similarity Filtering

Length Filtering [Arasu et al., 2006]

- Similar records have similar lengths
- E.g.
  - $sim$ is Jaccard
  - $\tau = .8$
  - Record length is 5
- Similar records have length $\in [4, 6]$
Length Filtering [Arasu et al., 2006]

- Similar records have similar lengths
- E.g.
  - \( sim \) is Jaccard
  - \( \tau = 0.8 \)
  - Record length is 5
- Similar records have length \( \in [4, 6] \)
Set-Similarity Filtering

Length Filtering [Arasu et al., 2006]

- Similar records have similar lengths
- E.g.
  - $sim$ is Jaccard
  - $\tau = .8$
  - Record length is 5
- Similar records have length $\in [4, 6]$
Set-Similarity Filtering

Length Filtering [Arasu et al., 2006]

- Similar records have similar lengths
- E.g.
  - \( sim \) is Jaccard
  - \( \tau = 0.8 \)
  - Record length is 5
- Similar records have length \( \in [4, 6] \)
Set-Similarity Filtering

Position Filtering [Xiao et al., 2008]

- Global order for set elements

\[
\text{Jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|}
\]

Maximum Jaccard(10, 20) = \frac{4}{6} = 0.66 < 0.8
Set-Similarity Filtering

Position Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., \( \text{sim} \) is Jaccard, \( \tau = .8 \)

\[
\text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|}
\]

\[
\text{maximum jaccard}(10, 20) = \frac{4}{6} = .66 < .8
\]
Set-Similarity Filtering

Position Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is Jaccard, $\tau = .8$
- Prefix length is 2

\[
\text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|}
\]

Maximum jaccard($10$, $20$) = \frac{4}{6} = .66 < .8
Set-Similarity Filtering

Position Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., sim is Jaccard, $\tau = .8$
- Prefix length is 2

\[
\text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|}
\]

\[\text{maximum jaccard}(10, 20) = \frac{4}{6} = 0.66 < 0.8\]
Set-Similarity Filtering

Position Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is Jaccard, $\tau = .8$
- Prefix length is 2

\[
\text{max } |n| = 1 \quad 3 \quad = 4
\]

\[
\text{jaccard}(x, y) = \frac{|x \cap y|}{|x \cup y|}
\]

\[
\text{maximum } \text{jaccard}(10, 20) = \frac{4}{6} = .66
\]
Set-Similarity Filtering

Position Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is Jaccard, $\tau = .8$
- Prefix length is 2

$$jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

- maximum $jaccard(10, 20) = \frac{4}{6} = .66 < .8$
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements

Prefix length is 2

E.g., $\text{sim}$ is overlap size, $\tau = 7$
Set-Similarity Filtering

**Suffix Filtering [Xiao et al., 2008]**

- Global order for set elements
- E.g., $sim$ is overlap size, $\tau = 7$

```plaintext
10  
20  
```
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is overlap size, $\tau = 7$
- Prefix length is 2

10  

20  

Prefix length is 2
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is overlap size, $\tau = 7$
- Prefix length is 2

10

20
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., sim is overlap size, $\tau = 7$
- Prefix length is 2

![Diagram showing suffix filtering]

- Max $|n| = 8$
  - $10$ and $20$

Rares Vernica (UC Irvine)
Set-Similarity Filtering

### Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., \( \text{sim} \) is overlap size, \( \tau = 7 \)
- Prefix length is 2

<table>
<thead>
<tr>
<th>10</th>
<th>[ \text{Hexagon}, \text{Triangle}, \text{Square}, \text{Circle}, \text{Star}, \text{Red Pentagon}, \text{Yellow Pentagon}, \text{Gray} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>[ \text{Hexagon}, \text{Triangle}, \text{Square}, \text{Circle}, \text{Star}, \text{Red Pentagon}, \text{Yellow Pentagon}, \text{Gray} ]</td>
</tr>
</tbody>
</table>

\[ \max |n| = 2 \quad 6 \quad = 8 \]
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., \( \text{sim} \) is overlap size, \( \tau = 7 \)
- Prefix length is 2

\[
\begin{align*}
10 & \quad \text{max } |n| = 2 \\
20 & \quad \text{max } |n| = 6 \\
\end{align*}
\]

\( \Rightarrow \) \( = 8 \)
Global order for set elements
E.g., $sim$ is overlap size, $\tau = 7$
Prefix length is 2

$max |n| = 8$
Set-Similarity Filtering

**Suffix Filtering [Xiao et al., 2008]**

- Global order for set elements
- E.g., $sim$ is overlap size, $\tau = 7$
- Prefix length is 2

![Diagram showing suffix filtering example](image)

Rares Vernica (UC Irvine)  
Fuzzy-Joins in MapReduce
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is overlap size, $\tau = 7$
- Prefix length is 2

Example:

10

20

$max |n| = 7$
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., \( \text{sim} \) is overlap size, \( \tau = 7 \)
- Prefix length is 2

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\text{10} & \text{10} & \text{20} & \text{20} \\
\hline
\text{max} |n| & 2 & 1 & 1 & 3 & 7
\end{array}
\]
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $sim$ is overlap size, $\tau = 7$
- Prefix length is 2

Global order for set elements

E.g., $sim$ is overlap size, $\tau = 7$

Prefix length is 2
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., \( sim \) is overlap size, \( \tau = 7 \)
- Prefix length is 2

\[
\begin{align*}
10 & \quad \begin{array}{cccccc}
\triangle & \triangle & \tau & \tau & \tau & \tau \\
\end{array} \\
20 & \quad \begin{array}{cccccc}
\triangle & \triangle & \tau & \tau & \tau & \tau \\
\end{array}
\end{align*}
\]

\[
\text{max } |n| = 2 \quad 6 \quad = 8
\]

\( |n| \geq 7 \)
Set-Similarity Filtering

Suffix Filtering [Xiao et al., 2008]

- Global order for set elements
- E.g., $\text{sim}$ is overlap size, $\tau = 7$
- Prefix length is 2

$\text{max } |n| = 7$

Rares Vernica (UC Irvine)
Outline

1 Motivation

2 Problem Statement

3 Single Machine Algorithms
   - Inverted List Index
   - Set-Similarity Filters

4 Parallel Algorithms

5 Experimental Evaluation
Parallelizing Set-Similarity Joins

Large amounts of data
- E.g., GeneBank: 100M, Google N-gram: 1T
- Data or processing does not fit in one machine
- Use a cluster of machines and a parallel algorithm
- **MapReduce**: shared-nothing data-processing platform

Challenges
- Partition problem for parallelism
- Solve using **Map, Sort, and Reduce**
- Compute **end-to-end** set-similarity joins
- Deal with out-of-memory situations
Parallelizing Set-Similarity Joins

**Large amounts of data**
- E.g., GeneBank: 100M, Google N-gram: 1T
- Data or processing does not fit in one machine
- Use a cluster of machines and a parallel algorithm
- **MapReduce**: shared-nothing data-processing platform

**Challenges**
- Partition problem for parallelism
- Solve using **Map, Sort, and Reduce**
- Compute *end-to-end* set-similarity joins
- Deal with out-of-memory situations
map \ (k_1,v_1) \rightarrow \text{list}(k_2,v_2);
reduce \ (k_2,\text{list}(v_2)) \rightarrow \text{list}(k_3,v_3).

combine \ (k_2,\text{list}(v_2)) \rightarrow \text{list}(k_2,v_2).
map \ (k1,v1) \rightarrow \text{list} (k2,v2); 
reduce \ (k2,\text{list} (v2)) \rightarrow \text{list} (k3,v3).

combine \ (k2,\text{list} (v2)) \rightarrow \text{list} (k2,v2).
map \ ((k_1, v_1)) \rightarrow \text{list}(k_2, v_2);
reduce \ ((k_2, \text{list}(v_2))) \rightarrow \text{list}(k_3, v_3).

combine \ ((k_2, \text{list}(v_2))) \rightarrow \text{list}(k_2, v_2).
Parallel Set-Similarity Joins in MapReduce

Main idea
- Hash-partition data across the network based on keys
- Join values **cannot** be directly used as keys
- Use set tokens as keys
  - e.g., “John W Smith” → \( \{\text{John, Smith, W}\} \)

Partition using prefix filter
- Use tokens in the prefix as keys
  - Minimize replication
- Global order: increasing frequency
  - Reduce skew
Parallel Set-Similarity Joins in MapReduce

Main idea
- Hash-partition data across the network based on keys
- Join values **cannot** be directly used as keys
- Use set tokens as keys
  - e.g., “John W Smith” \( \rightarrow \{ \text{John, Smith, W} \} \)

Partition using prefix filter
- Use tokens in the prefix as keys
  - **Minimize replication**
- Global order: increasing frequency
  - **Reduce skew**
Processing Stages and Alternatives

**Stage 1: Token Ordering**
- Compute the token frequencies and sort
  \[\text{Records} \rightarrow \text{Token\_Order}\]

**Stage 2: Kernel (RID-Pair Generation)**
- Generate similar record-ID ("RID") pairs
  \[\{\text{Records, Token\_Order}\} \rightarrow \text{RID-pairs}\]

**Stage 3: Record Join**
- Generate pairs of joined records
  \[\{\text{Records, RID-pairs}\} \rightarrow \text{Record-pairs}\]
Processing Stages and Alternatives

Stage 1: Token Ordering
- Compute the token frequencies and sort
  \[ \text{Records} \rightarrow \text{Token\_Order} \]

Stage 2: Kernel (RID-Pair Generation)
- Generate similar record-ID ("RID") pairs
  \[ \{\text{Records, Token\_Order}\} \rightarrow \text{RID\-pairs} \]

Stage 3: Record Join
- Generate pairs of joined records
  \[ \{\text{Records, RID\-pairs}\} \rightarrow \text{Record\-pairs} \]
Processing Stages and Alternatives

Stage 1: Token Ordering
- Compute the token frequencies and sort
  
  Records $\rightarrow$ Token_Order

Stage 2: Kernel (RID-Pair Generation)
- Generate similar record-ID ("RID") pairs
  
  \{Records, Token_Order\} $\rightarrow$ RID-pairs

Stage 3: Record Join
- Generate pairs of joined records
  
  \{Records, RID-pairs\} $\rightarrow$ Record-pairs
Stage 1: Token Ordering

Overview

- **Input:** original records
- **Output:** token order
- Compute the token frequencies and sort

Alternatives

- Basic Token Ordering (BTO)
  - Two MapReduce phases: sort in MapReduce
- One Phase Token Ordering (OPTO)
  - One MapReduce phase: sort in memory
Stage 1: Basic Token Ordering (BTO)
### Stage 1: Basic Token Ordering (BTO)

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
<th>Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td></td>
<td>A 1</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td></td>
<td>B 1</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td></td>
<td>C 2</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td></td>
<td>F 1</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td></td>
<td>G 1</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Map** Stage: Basic Token Ordering

- **Compute token frequencies**
- **Group by key**

**Reduce** Stage:

- **Sort tokens by frequency**
- **Group by key**

Rares Vernica (UC Irvine)

Fuzzy-Joins in MapReduce
## Stage 1: Basic Token Ordering (BTO)

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
<th>Key Value</th>
<th>Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td>...</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td>...</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td>...</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td>...</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td>...</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td>...</td>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

### Phase 1
- **Map**
  - Token frequencies are computed.

### Phase 2
- **Reduce**
  - Tokens are sorted by frequency.
- **Group by key**
  - Data is grouped by key.

The diagram illustrates the process of mapping and reducing tokens to achieve basic token ordering.
Stage 1: Basic Token Ordering (BTO)

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Map

Key Value

A 1
B 1
... 1

Group by key

Reduce

B 2
... 2

Reduce

A 2
... 4

Reduce

C 3
... 2

Phase 1: Compute token frequencies
Stage 1: Basic Token Ordering (BTO)

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td>...</td>
</tr>
</tbody>
</table>

Phase 1: Compute token frequencies
Stage 1: Basic Token Ordering (BTO)

Phase 1: Compute token frequencies

- **Map**
  - RID: 1, 2, 10, 11, 20, 21
  - a, b: A, B, C, ..., D, E, F

- **Group by key**
  - Key Value: A 1, B 1, C 1

- **Reduce**
  - Key Value: B 1, C 1

Phase 2: Sort tokens by frequency

- **Map**
  - Key Value: A 1, B 1

- **Group by key**
  - Key Value: B 2

- **Reduce**
  - Key Value: A 2

- **Map**
  - Key Value: C 3

- **Group by key**
  - Key Value: C 2

- **Reduce**
  - Key Value: E 2

- **Map**
  - Key Value: E 2

- **Group by key**
  - Key Value: C 2

- **Reduce**
  - Key Value: F 4

- **Map**
  - Key Value: E 2

- **Group by key**
  - Key Value: A 2, B 2, C 2, D 2, E 2, F 4
Stage 1: Basic Token Ordering (BTO)

Phase 1: Compute token frequencies

Phase 2: Sort tokens by frequency
Stage 1: One Phase Token Ordering (OPTO)

Rares Vernica (UC Irvine)
Stage 2: Kernel (RID-Pair Generation)

Overview

- **Input**: original records and token order
- **Output**: list of similar-RID pairs
- Partition using prefix filter

Steps

- Load token order in memory
- Extract RIDs and join value of records
- Distribute records on prefix tokens
- Group RIDs and join values
- Cross-pair and verify candidates
Stage 2: Kernel (RID-Pair Generation)

Overview

- **Input:** original records and token order
- **Output:** list of similar-RID pairs
- Partition using prefix filter

Steps

- Load token order in memory
  - Extract RIDs and join value of records
  - Distribute records on prefix tokens
  - Group RIDs and join values
  - Cross-pair and verify candidates
Stage 2: Kernel (RID-Pair Generation)

Overview
- **Input:** original records and token order
- **Output:** list of similar-RID pairs
- Partition using prefix filter

Steps
- Load token order in memory
- Extract RIDs and join value of records
- Distribute records on prefix tokens
- Group RIDs and join values
- Cross-pair and verify candidates

Map
## Stage 2: Kernel (RID-Pair Generation)

### Overview
- **Input:** original records and token order
- **Output:** list of similar-RID pairs
- Partition using prefix filter

### Steps
- Load token order in memory
- Extract RIDs and join value of records
- Distribute records on prefix tokens
  - Group RIDs and join values
  - Cross-pair and verify candidates
  - Map
  - Shuffle

---

Rares Vernica (UC Irvine)  
Fuzzy-Joins in MapReduce
Stage 2: Kernel (RID-Pair Generation)

Overview
- **Input:** original records and token order
- **Output:** list of similar-RID pairs
- Partition using prefix filter

Steps
- Load token order in memory
- Extract RIDs and join value of records
- Distribute records on prefix tokens
- Group RIDs and join values
- Cross-pair and verify candidates

Map
Shuffle
Stage 2: Kernel (RID-Pair Generation)

Overview
- **Input**: original records and token order
- **Output**: list of similar-RID pairs
- Partition using prefix filter

Steps
- Load token order in memory
- Extract RIDs and join value of records
- Distribute records on prefix tokens
- Group RIDs and join values
- Cross-pair and verify candidates

\{ Map Shuffle \} Reduce
Stage 2: Partition Using Individual Tokens

```
Token

G ...

Map

Map

Map
```

Rares Vernica  (UC Irvine)  Fuzzy-Joins in MapReduce
Stage 2: Partition Using Individual Tokens

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Stage 2: Partition Using Individual Tokens

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>CF</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>ECD</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>FG</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>BAF</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Token

RID       a        b
1 A B C    ...     
2 D E F    ...     
... ...     ...     
10 C F     ...     
11 E C D   ...     
... ...     ...     
20 F G     ...     
21 B A F   ...     
... ...     ...     

Key     Value

Group by key

1 A B C  
21 B A F  
... ...           

Reduce

RID1 RID2 Sim.
0.5
... 
0.5
... 
21
... 
11
... 
... 
... 
... 
...
Stage 2: Partition Using Individual Tokens

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Map**

  - **Token**: G
  - **Key**: A
  - **Value**: 1, A B C
  - **Value**: 1, A B C

  - **Key**: B
  - **Value**: 1, A B C
  - **Value**: 1, A B C

  - **Key**: C
  - **Value**: 10, C F
  - **Value**: 11, E C D

  - **Key**: D
  - **Value**: 10, C F
  - **Value**: 11, E C D

  - **Key**: G
  - **Value**: 20, F G
  - **Value**: 21, B A F

  - **Key**: A
  - **Value**: 20, F G
  - **Value**: 21, B A F

- **Reduce**

  - **RID**: 1, RID: 2
  - **Similarity**: 0.5
  - **Similarity**: 0.5
  - **Similarity**: 0.5
  - **Similarity**: 0.5
  - **Similarity**: 0.5
Stage 2: Partition Using Individual Tokens

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td></td>
<td>A</td>
<td>1, A B C</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td></td>
<td>B</td>
<td>1, A B C</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td></td>
<td>C</td>
<td>10, C F</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td></td>
<td>D</td>
<td>11, E C D</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td></td>
<td>G</td>
<td>20, F G</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td></td>
<td>A</td>
<td>21, B A F</td>
</tr>
</tbody>
</table>

Map

Group by key

Key | Value
B   | 1, A B C
B   | 21, B A F
A   | 1, A B C
A   | 21, B A F
C   | 10, C F
E   | 2, D E F
Stage 2: Partition Using Individual Tokens

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td></td>
<td>A</td>
<td>1, A B C</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td></td>
<td>B</td>
<td>1, A B C</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td></td>
<td>C</td>
<td>10, C F</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td></td>
<td>D</td>
<td>11, E C D</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td></td>
<td>G</td>
<td>20, F G</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td></td>
<td>A</td>
<td>21, B A F</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Group by key

Rares Vernica (UC Irvine)
Stage 2: Partition Using Individual Tokens

Alternatives
- Basic Kernel (BK): nested loops
- PPJoin+ Kernel (PK): inverted list index

```
<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Map

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1, A B C</td>
</tr>
<tr>
<td>B</td>
<td>1, A B C</td>
</tr>
<tr>
<td>C</td>
<td>10, C F</td>
</tr>
<tr>
<td>D</td>
<td>11, E C D</td>
</tr>
<tr>
<td>G</td>
<td>20, F G</td>
</tr>
<tr>
<td>A</td>
<td>21, B A F</td>
</tr>
<tr>
<td>B</td>
<td>21, B A F</td>
</tr>
<tr>
<td>A</td>
<td>21, B A F</td>
</tr>
<tr>
<td>E</td>
<td>2, D E F</td>
</tr>
</tbody>
</table>
```

Group by key

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1, A B C</td>
</tr>
<tr>
<td>B</td>
<td>21, B A F</td>
</tr>
<tr>
<td>A</td>
<td>1, A B C</td>
</tr>
<tr>
<td>A</td>
<td>21, B A F</td>
</tr>
<tr>
<td>C</td>
<td>10, C F</td>
</tr>
<tr>
<td>E</td>
<td>2, D E F</td>
</tr>
</tbody>
</table>
```

Reduce

```
<table>
<thead>
<tr>
<th>RID1</th>
<th>RID2</th>
<th>Sim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Stage 2: Partition Using Grouped Tokens

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
<th>RID1</th>
<th>RID2</th>
<th>Sim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td>...</td>
<td>X</td>
<td>1,A B C</td>
<td>Y</td>
<td>1,A B C</td>
<td>1</td>
<td>21</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td>...</td>
<td>Y</td>
<td>1,A B C</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td>...</td>
<td>Z</td>
<td>10,C F</td>
<td>Z</td>
<td>11,E C D</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td>...</td>
<td>Y</td>
<td>20,F G</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td>...</td>
<td>X</td>
<td>21,B A F</td>
<td>...</td>
<td>...</td>
<td>2</td>
<td>11</td>
<td>0.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Token Grouping Func.
Stage 3: Record Join

Overview

- **Input**: original records and similar-RID pairs
- **Output**: similar-record pairs
- Generate pairs of similar records

Alternatives

- Basic Record Join (BRJ)
  - Two MapReduce phases: reduce-side join
- One Phase Record Join (OPRJ)
  - One MapReduce phase: map-side join
Additional Contributions

- Solve R-S-join case
- Optimize memory requirements for R-S join
- Handle insufficient memory
  - Introduce additional filters
  - Use external memory
Outline

1. Motivation

2. Problem Statement

   - Inverted List Index
   - Set-Similarity Filters

4. Parallel Algorithms

5. Experimental Evaluation
Experimental Setting

Hardware
- 10-node IBM x3650 cluster
  - Intel Xeon processor E5520 2.26GHz with four cores
  - Four 300GB hard disks
  - 12GB RAM

Software
- Ubuntu 9.06, 64-bit, server edition OS
- Java 1.6, 64-bit, server
- Hadoop 0.20.1
Experimental Setting

Datasets

- **DBLP**
  - Average length: 259 bytes
  - Number of records: 1.2M
  - Total size: 300MB

- **CITESEERX**
  - Average length: 1374 bytes
  - Number of records: 1.3M
  - Total size: 1.8GB

- Increased each up to $\times25$, preserving join properties
  - DBLP: 31M records, 8.2GB
  - CITESEERX: 32M records, 45GB
Running Time

- Self-join DBLP $\times n$
- $n \in [5, 25]$
- 10-node cluster
- Best time
- Bulk of the time

Dataset Size (times the original)

Time (seconds)
Running Time

Legend
- Stage 1
  - BTO: Basic Token Ordering
  - OPTO: One Phase Token Ordering
- Stage 2
  - BK: Basic Kernel
  - PK: PPJoin+ Kernel
- Stage 3
  - BRJ: Basic Record Join
  - OPRJ: One Phase Record Join
Running Time

- Self-join DBLP $\times n$
- $n \in [5, 25]$
- 10-node cluster
- **Best time**
- Bulk of the time

Time (seconds)

Dataset Size (times the original)
Running Time

- Self-join DBLP $\times n$
- $n \in [5, 25]$
- 10-node cluster
- Best time
- Bulk of the time
Speedup

- BTO-BK-BRJ
- BTO-PK-BRJ
- BTO-PK-OPRJ
- Ideal

Relative running time
Self-join DBLP × 10
Different cluster sizes

Rares Vernica (UC Irvine)
Fuzzy-Joins in MapReduce
Speedup Breakdown

Stage 1

Stage 2

Stage 3

- Relative running time
- Self-join DBLP × 10
- Different cluster sizes
Scaleup

Running time
Self-joining DBLP $\times n$
$n \in [5, 25]$
Proportional cluster

- BTO-BK-BRJ
- BTO-PK-BRJ
- BTO-PK-OPRJ
- Ideal

Rares Vernica (UC Irvine)
Fuzzy-Joins in MapReduce
Scaleup Breakdown

- **Running time**
- **Self-joining DBLP \( \times n, n \in [5, 25] \)**
- **Proportional cluster**
Summary

- Set-similarity joins in MapReduce
  - Three-stage approach
  - Balance workload and minimize replication
- *End-to-end* algorithms
  - Self-join
  - R-S join
- Memory issues
  - Optimize memory requirements
  - Handle insufficient memory
- Experiments
  - Speedup and scaleup
  - 40 cores, 40 disks cluster
The paper, the source-code and the datasets:
http://asterix.ics.uci.edu/fuzzyjoin-mapreduce/

This work is part of the
ASTERIX
http://asterix.ics.uci.edu/
and
Flamingo
http://flamingo.ics.uci.edu/
projects at UC Irvine.


### Stage 3: Basic Record Join (BRJ)

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>ABC</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>DEF</code></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td><code>CF</code></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td><code>ECD</code></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RID1</th>
<th>RID2</th>
<th>Sim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>0.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Map**

- `1,A B C,...`
- `2,D E F,...`
- `...`
- `10,C F,...`
- `11,E C D,...`
- `...`
- `(2,11),0.5`
- `...`

**Key**

**Value**

- Entire Record

**Group by key**

- `2,D E F,...`
- `(2,11),0.5`
- `...`
- `1,A B C,...`
- `(1,21),0.5`
- `...`
- `11,E C D,...`
- `(2,11),0.5`
- `...`

**Reduce**

- `2,11`
- `1,21`
- `...`
- `...`
- `2,D E F,...,0.5`
- `21,B A F,...,0.5`
- `...`
- `1,A B C,...,0.5`
- `...`
- `11,E C D,...,0.5`
- `...`
Stage 3: Basic Record Join (BRJ)

RID | a | b
--- | --- | ---
1   | A B C | ...
2   | D E F | ...
10  | C F   | ...
11  | E C D | ...

Map

Key | Value
--- | ---
1   | A B C,..
2   | D E F,..
10  | C F,..
11  | E C D,..

 Entire Record

Phase 1: Duplicate the RID pairs and fill half on each

RID1 | RID2 | Sim.
--- | --- | ---
2   | 1   | 0.5
1   | 21  | 0.5

Map

Key | Value
--- | ---
2   | (2,11),0.5
11  | (2,11),0.5
Stage 3: Basic Record Join (BRJ)

Duplicate the RID pairs and fill half on each.

Phase 1

Rares Vernica (UC Irvine)
Stage 3: Basic Record Join (BRJ)

Phase 1: Duplicate the RID pairs and fill half on each

Map

RID | a   | b     |
--- | ---- | ------|
1   | A B C| ...   |
2   | D E F| ...   |
10  | C F  | ...   |
11  | E C D| ...   |
2   | 11   | 0.5   |
1   | 21   | 0.5   |

Key | Value
--- | ---
1   | 1,A B C,..., 0.5
2   | 2,D E F,..., 0.5
10  | 10,C F,..., 0.5
11  | 11,E C D,..., 0.5
2   | (2,11),0.5
11  | (2,11),0.5

Entire Record

Group by key

Map

Key | Value
--- | ---
1   | 1,A B C,..., 0.5
2   | 2,D E F,..., 0.5
11  | 11,E C D,..., 0.5
2   | (2,11),0.5

Reduce

Key | Value
--- | ---
1   | 1,A B C,..., 0.5
2   | 2,D E F,..., 0.5
11  | 11,E C D,..., 0.5
2   | (2,11),0.5

Reduce

Entire Record
Stage 3: Basic Record Join (BRJ)

Identity Map

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,11</td>
<td>2,D E F,...,0.5</td>
</tr>
<tr>
<td>1,21</td>
<td>21,B A F,...,0.5</td>
</tr>
<tr>
<td></td>
<td>1,A B C,...,0.5</td>
</tr>
<tr>
<td></td>
<td>11,E C D,...,0.5</td>
</tr>
</tbody>
</table>

Group by key

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,11</td>
<td>2,D E F,...,0.5</td>
</tr>
<tr>
<td>1,21</td>
<td>21,B A F,...,0.5</td>
</tr>
<tr>
<td></td>
<td>1,A B C,...,0.5</td>
</tr>
<tr>
<td></td>
<td>11,E C D,...,0.5</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>RID1</th>
<th>a1</th>
<th>b1</th>
<th>Sim.</th>
<th>RID2</th>
<th>a2</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
<td>11</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E C D</td>
<td>B A F</td>
<td></td>
</tr>
</tbody>
</table>

Bring together and fill-in the half filled pairs
Stage 3: Basic Record Join (BRJ)

Identity Map

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,11</td>
<td>2, D E F, ..., 0.5</td>
</tr>
<tr>
<td>1,21</td>
<td>21, B A F, ..., 0.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Group by key

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,21</td>
<td>1, A B C, ..., 0.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,11</td>
<td>21, B A F, ..., 0.5</td>
</tr>
<tr>
<td>11</td>
<td>11, E C D, ..., 0.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>RID1</th>
<th>a1</th>
<th>b1</th>
<th>Sim.</th>
<th>RID2</th>
<th>a2</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>...</td>
<td>...</td>
<td></td>
<td>0.5</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0.5</td>
<td>...</td>
<td>...</td>
<td></td>
<td>0.5</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Phase 2

Bring together and fill-in the half filled pairs
Stage 3: Basic Record Join (BRJ)

Identity Map

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,11</td>
<td>2, D E F, ..., 0.5</td>
</tr>
<tr>
<td>1,21</td>
<td>21, B A F, ..., 0.5</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Group by key

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,21</td>
<td>1, A B C, ..., 0.5</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,11</td>
<td>2, D E F, ..., 0.5</td>
</tr>
<tr>
<td>1,21</td>
<td>1, A B C, ..., 0.5</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Reduce

<table>
<thead>
<tr>
<th>RID</th>
<th>a1</th>
<th>b1</th>
<th>Sim.</th>
<th>RID</th>
<th>a2</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>0.5</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Phase 2
Bring together and fill-in the half filled pairs

Rares Vernica (UC Irvine)  Fuzzy-Joins in MapReduce
Stage 3: One-Phase Record Join (OPRJ)

RID1 RID2 Sim.

2 11 0.5
...
...
...

Map

Map

Map

Map
Stage 3: One-Phase Record Join (OPRJ)

1. Map

   - RID1 RID2 Sim.
   - 2   11  0.5
   - ... ...

2. Reduce

   - RID1     a1   b1   Sim.  RID2     a2   b2
   - 1   A B C  ...   2   D E F  ...
   - 2   D E F  ...   2   D E F  ...
   - ... ...    ...   ... ...
   - 10  C F    ...  11  E C D ...
   - ... ...    ...   ... ...
   - 20  F G    ...  21  B A F ...
   - ... ...    ...   ... ...

Rares Vernica (UC Irvine)
Fuzzy-Joins in MapReduce
Stage 3: One-Phase Record Join (OPRJ)

The diagram illustrates the process of a one-phase record join in MapReduce, where records are grouped and reduced to form a final set of joined records. The process involves mapping records to keys based on predefined conditions and then aggregating values associated with those keys.

Key:
- **Map**: Records are mapped to keys based on predefined conditions.
- **Reduce**: Aggregates values associated with those keys.

Value:
- Entire Record

### Examples

#### RID1 RID2 Sim.
- RID 2, 11, 0.5
- Entire Record

#### RID a b

<table>
<thead>
<tr>
<th>RID</th>
<th>a</th>
<th>b</th>
<th>Sim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A B C</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D E F</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C F</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>E C D</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>F G</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>B A F</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

MapReduce process:
- Records are mapped to keys.
- Values are aggregated for each key.
- The final set of joined records is produced.

Rares Vernica (UC Irvine)
Stage 3: One-Phase Record Join (OPRJ)

```
Rides Vernica (UC Irvine)
Fuzzy-Joins in MapReduce
```
Stage 3: One-Phase Record Join (OPRJ)
Memory Requirement Optimizations (self-join)

- Group by *Token* and sort by *(Token, Length)*
- Keep in memory only the records within the length range

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>90</td>
</tr>
</tbody>
</table>
Memory Requirement Optimizations (self-join)

- Group by *Token* and sort by (*Token*, *Length*)
- Keep in memory only the records within the length range

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
Memory Requirement Optimizations (self-join)

- Group by \textit{Token} and sort by (\textit{Token}, \textit{Length})
- Keep in memory only the records within the length range

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
Memory Requirement Optimizations (self-join)

- Group by *Token* and sort by (*Token, Length*)
- Keep in memory only the records within the length range

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
</tbody>
</table>
Memory Requirement Optimizations (R-S join)

- Group by *Token* and sort by (*Token, Length, Relation*)
- For *R* sort by (*Token, Relation, Length'*)

<table>
<thead>
<tr>
<th>R10</th>
<th>Length</th>
<th>S10</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>R20</td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>R30</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>R40</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>R50</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>R60</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>R70</td>
<td>6</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>R80</td>
<td>6</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>R90</td>
<td>7</td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S50</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S60</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S70</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S80</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S90</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>
Memory Requirement Optimizations (R-S join)

- Group by *Token* and sort by *(Token, Length, Relation)*
- For *R* sort by *(Token, Relation, Length')*

<table>
<thead>
<tr>
<th></th>
<th>Length</th>
<th></th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>R10</td>
<td>3</td>
<td>S10</td>
<td>3</td>
</tr>
<tr>
<td>R20</td>
<td>3</td>
<td>S20</td>
<td>3</td>
</tr>
<tr>
<td>R30</td>
<td>4</td>
<td>S30</td>
<td>4</td>
</tr>
<tr>
<td>R40</td>
<td>4</td>
<td>S40</td>
<td>4</td>
</tr>
<tr>
<td>R50</td>
<td>5</td>
<td>S50</td>
<td>5</td>
</tr>
<tr>
<td>R60</td>
<td>5</td>
<td>S60</td>
<td>5</td>
</tr>
<tr>
<td>R70</td>
<td>6</td>
<td>S70</td>
<td>6</td>
</tr>
<tr>
<td>R80</td>
<td>6</td>
<td>S80</td>
<td>6</td>
</tr>
<tr>
<td>R90</td>
<td>7</td>
<td>S90</td>
<td>7</td>
</tr>
</tbody>
</table>
Memory Requirement Optimizations (R-S join)

- Group by *Token* and sort by (*Token*, *Length*, *Relation*)
- For *R* sort by (*Token*, *Relation*, *Length'*)

<table>
<thead>
<tr>
<th>R10</th>
<th>R20</th>
<th>R30</th>
<th>R40</th>
<th>R50</th>
<th>R60</th>
<th>R70</th>
<th>R80</th>
<th>R90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Length: 3

<table>
<thead>
<tr>
<th>S10</th>
<th>S20</th>
<th>S30</th>
<th>S40</th>
<th>S50</th>
<th>S60</th>
<th>S70</th>
<th>S80</th>
<th>S90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Length: 3
Memory Requirement Optimizations (R-S join)

- Group by *Token* and sort by *(Token, Length, Relation)*
- For *R* sort by *(Token, Relation, Length')*

<table>
<thead>
<tr>
<th>Token</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>R10</td>
<td>3</td>
</tr>
<tr>
<td>R20</td>
<td>3</td>
</tr>
<tr>
<td>R30</td>
<td>4</td>
</tr>
<tr>
<td>R40</td>
<td>4</td>
</tr>
<tr>
<td>R50</td>
<td>5</td>
</tr>
<tr>
<td>R60</td>
<td>5</td>
</tr>
<tr>
<td>R70</td>
<td>6</td>
</tr>
<tr>
<td>R80</td>
<td>6</td>
</tr>
<tr>
<td>R90</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Token</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>S10</td>
<td>3</td>
</tr>
<tr>
<td>S20</td>
<td>3</td>
</tr>
<tr>
<td>S30</td>
<td>4</td>
</tr>
<tr>
<td>S40</td>
<td>4</td>
</tr>
<tr>
<td>S50</td>
<td>5</td>
</tr>
<tr>
<td>S60</td>
<td>5</td>
</tr>
<tr>
<td>S70</td>
<td>6</td>
</tr>
<tr>
<td>S80</td>
<td>6</td>
</tr>
<tr>
<td>S90</td>
<td>7</td>
</tr>
</tbody>
</table>
**Memory Requirement Optimizations (R-S join)**

- Group by *Token* and sort by (*Token*, *Length*, *Relation*)
- For *R* sort by (*Token*, *Relation*, *Length*')

<table>
<thead>
<tr>
<th>R10</th>
<th>R20</th>
<th>R30</th>
<th>R40</th>
<th>R50</th>
<th>R60</th>
<th>R70</th>
<th>R80</th>
<th>R90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>3 → 3</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>S10</th>
<th>S20</th>
<th>S30</th>
<th>S40</th>
<th>S50</th>
<th>S60</th>
<th>S70</th>
<th>S80</th>
<th>S90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>7</th>
</tr>
</thead>
</table>
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \Join A$
  - $A \Join B$
  - $A \Join C$
  - $B \Join B$
  - $B \Join C$
  - $C \Join C$
- Replicate data over network
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Replicate data over network
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Replicate data over network
Records that need to be cross-verified do not fit in memory

Example: split data in memory-size blocks $A$, $B$, and $C$

Need to compute:

- $A \bowtie A$
- $A \bowtie B$
- $A \bowtie C$
- $B \bowtie B$
- $B \bowtie C$
- $C \bowtie C$

Replicate data over network
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Replicate data over network
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Replicate data over network
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Replicate data over network
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \Join A$
  - $A \Join B$
  - $A \Join C$
  - $B \Join B$
  - $B \Join C$
  - $C \Join C$
- Spill data to disk
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Spill data to disk
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \Join A$
  - $A \Join B$
  - $A \Join C$
  - $B \Join B$
  - $B \Join C$
  - $C \Join C$
- Spill data to disk
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Spill data to disk
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$

- Spill data to disk

![Diagram showing data partitions and self-join operations with spill to disk](image-url)
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \Join A$
  - $A \Join B$
  - $A \Join C$
  - $B \Join B$
  - $B \Join C$
  - $C \Join C$

- Spill data to disk
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Spill data to disk
Using External Memory (self join)

- Records that need to be cross-verified do not fit in memory
- Example: split data in memory-size blocks $A$, $B$, and $C$
- Need to compute:
  - $A \bowtie A$
  - $A \bowtie B$
  - $A \bowtie C$
  - $B \bowtie B$
  - $B \bowtie C$
  - $C \bowtie C$
- Spill data to disk