

Efficient Parallel Set-Similarity Joins Using MapReduce

Rares Vernica Michael J. Carey Chen Li

Department of Computer Science
University of California, Irvine

Special Interest Group on Management of Data, 2010



Outline



1 Motivation

2 Problem Statement

3 Algorithms

4 Experimental Evaluation



Example: Data Cleaning/Master-Data-Management

Customer data from two departments

Sales

ID	Name	...
S10	John W Smith	...
⋮		

Returns

ID	Name	...
R20	Smith John	...
⋮		

Master customer data across two departments

Customers

ID	Name	...
C30	John W Smith	...
⋮		

Example: Data Cleaning/Master-Data-Management

Customer data from two departments

Sales

ID	Name	...
S10	John W Smith	...
⋮		

Returns

ID	Name	...
R20	Smith John	...
⋮		

Master customer data across two departments

Customers

ID	Name	...
C30	John W Smith	...
⋮		

Example: Data Cleaning/Master-Data-Management

Customer data from two departments

Sales

ID	Name	...
S10	John W Smith	...
⋮		

Returns

ID	Name	...
R20	Smith John	...
⋮		

Master customer data across two departments

Customers

ID	Name	...
C30	John W Smith	...
⋮		

Example: Data Cleaning/Master-Data-Management

Customer data from two departments

Sales		
ID	Name	...
S10	John W Smith	...
⋮		

Returns		
ID	Name	...
R20	John W Smith	...
⋮		

Master customer data across two departments

Customers		
ID	Name	...
C30	John W Smith	...
⋮		

Example: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S_{10} : "John W Smith" \rightarrow {John, Smith, W}
- R_{20} : "Smith John" \rightarrow {John, Smith}

Set-Similarity Metric

- Jaccard similarity/Tanimoto coefficient: $jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|}$
- $jaccard(S_{10}, R_{20}) = \frac{2}{3}$

Set-Similarity Join

Set-Similarity Join "Sales" and "Returns" on "Name"



Example: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S10: "John W Smith" \rightarrow {John, Smith, W}
- R20: "Smith John" \rightarrow {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: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S10: “John W Smith” \rightarrow {John, Smith, W}
- R20: “Smith John” \rightarrow {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: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S10: "John W Smith" \rightarrow {John, Smith, W}
- R20: "Smith John" \rightarrow {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: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S10: “John W Smith” \rightarrow {John, Smith, W}
- R20: “Smith John” \rightarrow {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: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S10: "John W Smith" \rightarrow {John, Smith, W}
- R20: "Smith John" \rightarrow {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: Data Cleaning/Master-Data-Management

String \rightarrow Set

- S10: "John W Smith" \rightarrow {John, Smith, W}
- R20: "Smith John" \rightarrow {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"

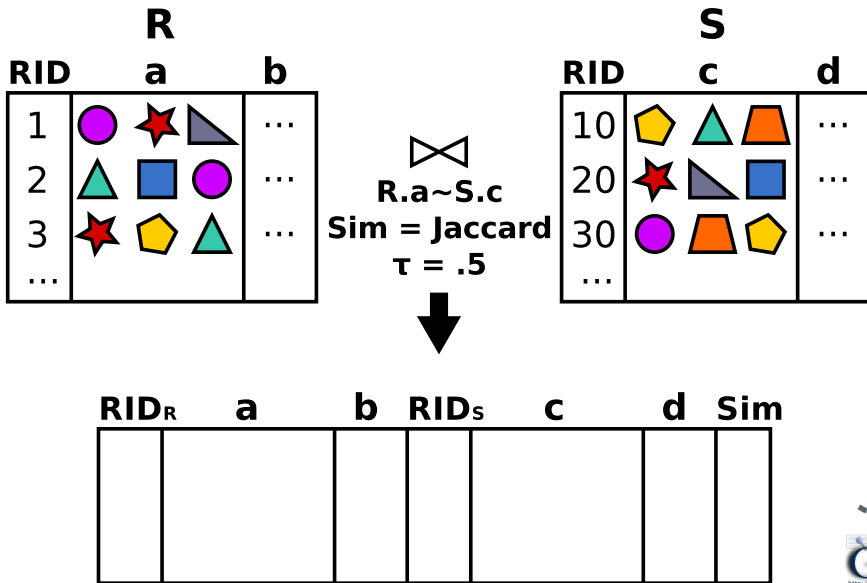


Problem Statement: Set-Similarity Join

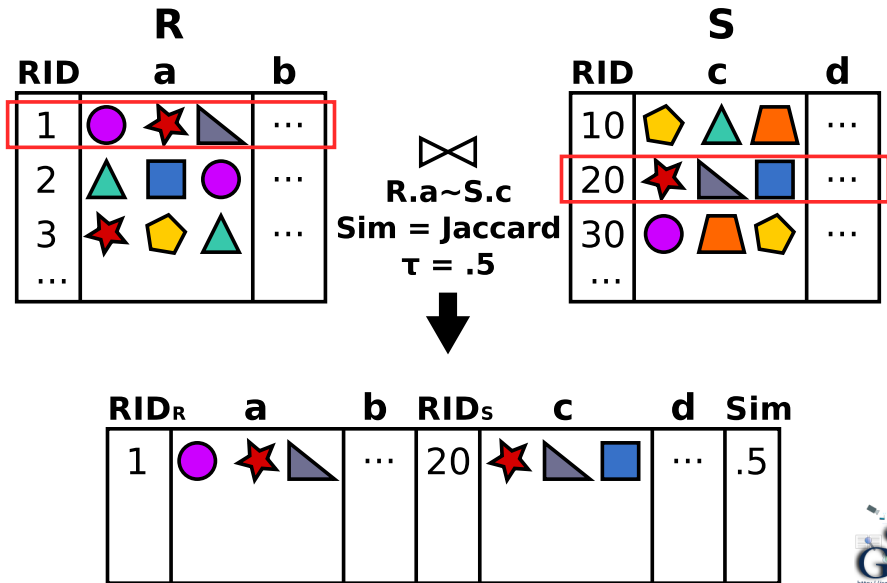
R				
RID	a	b		
1				...
2				...
3				...
...				

S				
RID	c	d		
10				...
20				...
30				...
...				

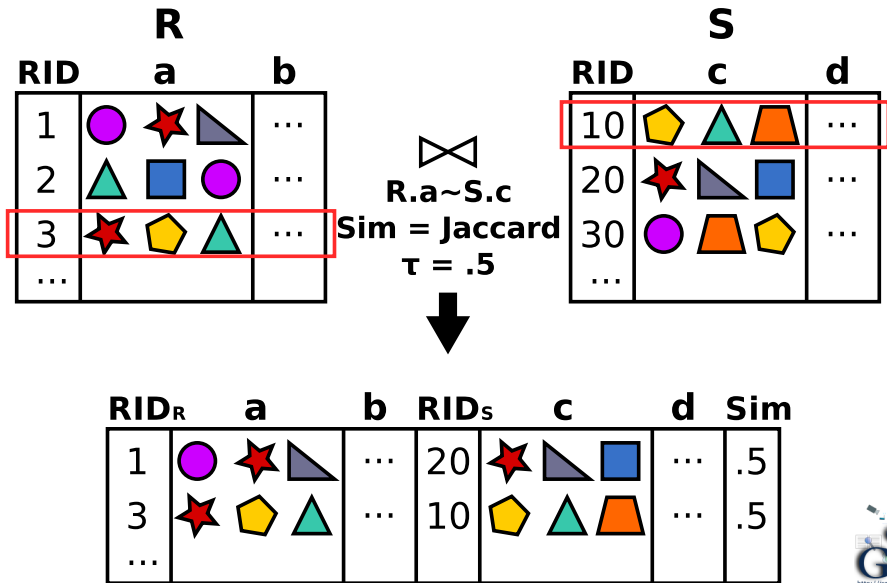
Problem Statement: Set-Similarity Join



Problem Statement: Set-Similarity Join



Problem Statement: Set-Similarity Join



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, τ

Output

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



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



MapReduce Review

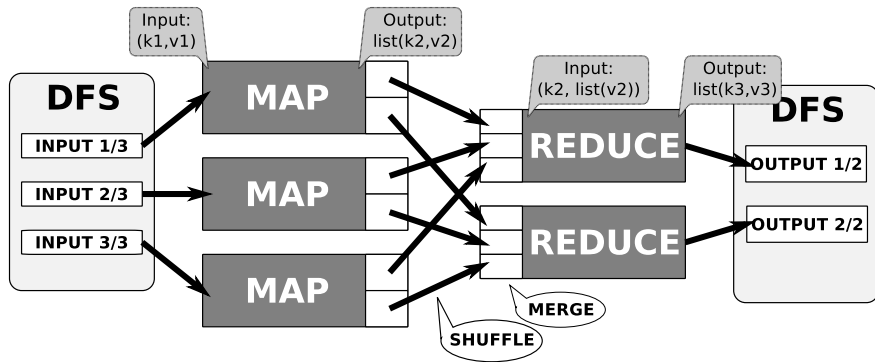
```
map      (k1, v1)      → list (k2, v2);  
reduce  (k2, list (v2)) → list (k3, v3).
```

```
combine (k2, list (v2)) → list (k2, v2).
```



MapReduce Review

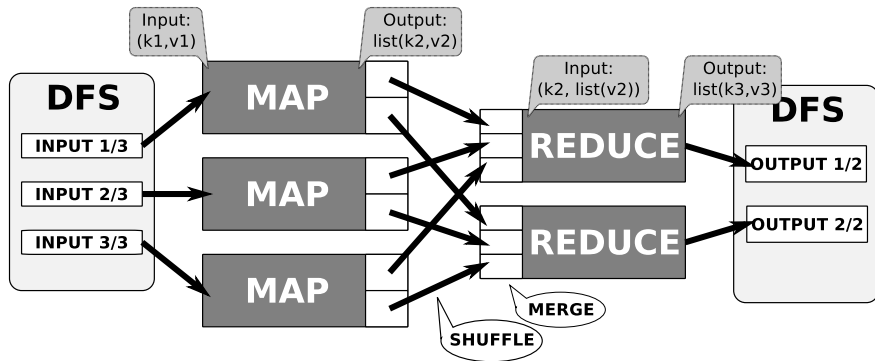
map (k1, v1) → list(k2, v2);
reduce (k2, list(v2)) → list(k3, v3).



combine (k2, list(v2)) → list(k2, v2).

MapReduce Review

map (k1, v1) → list(k2, v2);
reduce (k2, list(v2)) → list(k3, v3).



combine (k2, list(v2)) → list(k2, v2).

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
- Generate *signatures* from join values and use them as keys
 - e.g., “John W Smith” \rightarrow {John, Smith, W}

Three Stages

- 1 Compute set-element statistics
`Records \rightarrow Statistics`
- 2 Generate similar record-ID (“RID”) pairs
`{Records, Statistics} \rightarrow RID-pairs`
- 3 Generate pairs of joined records
`{Records, RID-pairs} \rightarrow Record-pairs`

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
- Generate *signatures* from join values and use them as keys
 - e.g., “John W Smith” \rightarrow {John, Smith, W}

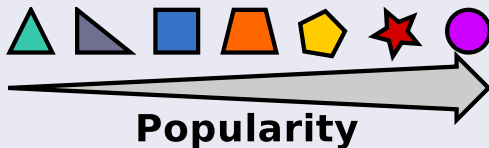
Three Stages

- 1 Compute set-element statistics
Records \rightarrow Statistics
- 2 Generate similar record-ID (“RID”) pairs
{Records, Statistics} \rightarrow RID-pairs
- 3 Generate pairs of joined records
{Records, RID-pairs} \rightarrow Record-pairs

Set-Similarity Filtering

E.g., Prefix Filtering

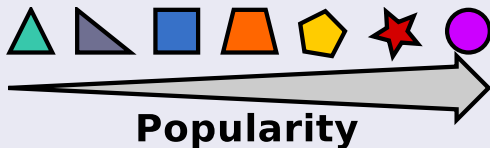
- Pigeonhole principle
- Global order for set elements:



Set-Similarity Filtering

E.g., Prefix Filtering

- Pigeonhole principle
- Global order for set elements:



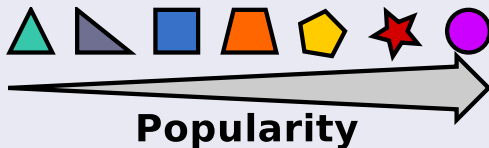
- E.g., sim is overlap size, $\tau = 4$



Set-Similarity Filtering

E.g., Prefix Filtering

- Pigeonhole principle
- Global order for set elements:



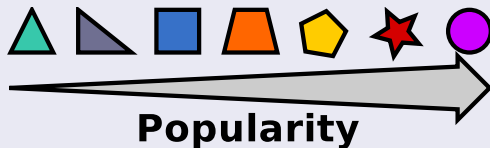
- E.g., sim is overlap size, $\tau = 4$
- Prefix length is 2



Set-Similarity Filtering

E.g., Prefix Filtering

- Pigeonhole principle
- Global order for set elements:



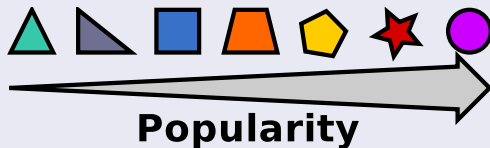
- E.g., sim is overlap size, $\tau = 4$
- Prefix length is 2



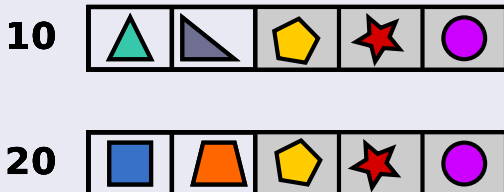
Set-Similarity Filtering

E.g., Prefix Filtering

- Pigeonhole principle
- Global order for set elements:



- E.g., sim is overlap size, $\tau = 4$
- Prefix length is 2



Processing Stages and Alternatives

Stage 1: Token Ordering

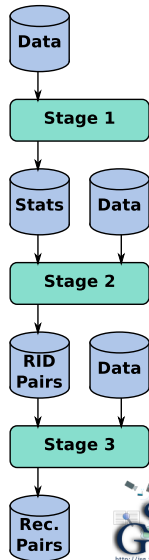
- Compute the token frequencies and sort
 - Two MapReduce phases: sort in MapReduce (BTO)
 - One MapReduce phase: sort in memory (OPTO)

Stage 2: Kernel (RID-Pair Generation)

- Use prefix-filter to *divide, conquer* using:
 - Nested loops (BK)
 - Single-machine set-similarity join algorithm (PK)

Stage 3: Record Join

- Generate pairs of similar records
 - Two MapReduce phases: reduce-side join (BRJ)
 - One MapReduce phase: map-side join (OPRJ)



Processing Stages and Alternatives

Stage 1: Token Ordering

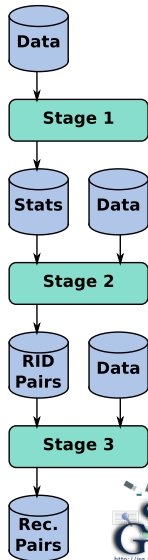
- Compute the token frequencies and sort
 - Two MapReduce phases: sort in MapReduce (BTO)
 - One MapReduce phase: sort in memory (OPTO)

Stage 2: Kernel (RID-Pair Generation)

- Use prefix-filter to *divide, conquer* using:
 - Nested loops (BK)
 - Single-machine set-similarity join algorithm (PK)

Stage 3: Record Join

- Generate pairs of similar records
 - Two MapReduce phases: reduce-side join (BRJ)
 - One MapReduce phase: map-side join (OPRJ)



Processing Stages and Alternatives

Stage 1: Token Ordering

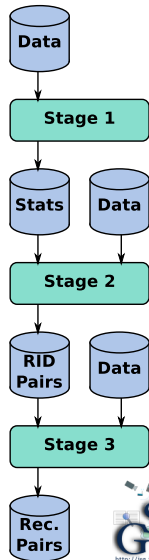
- Compute the token frequencies and sort
 - Two MapReduce phases: sort in MapReduce (BTO)
 - One MapReduce phase: sort in memory (OPTO)

Stage 2: Kernel (RID-Pair Generation)

- Use prefix-filter to *divide, conquer* using:
 - Nested loops (BK)
 - Single-machine set-similarity join algorithm (PK)

Stage 3: Record Join

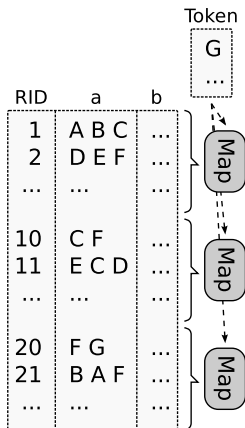
- Generate pairs of similar records
 - Two MapReduce phases: reduce-side join (BRJ)
 - One MapReduce phase: map-side join (OPRJ)



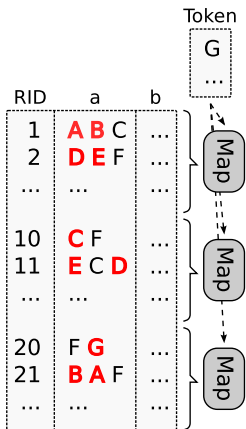
Stage 2: RID-Pair Generation



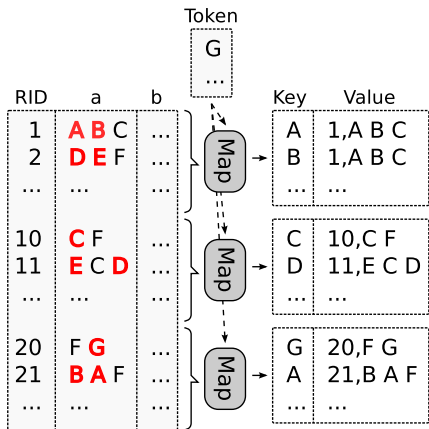
Stage 2: RID-Pair Generation



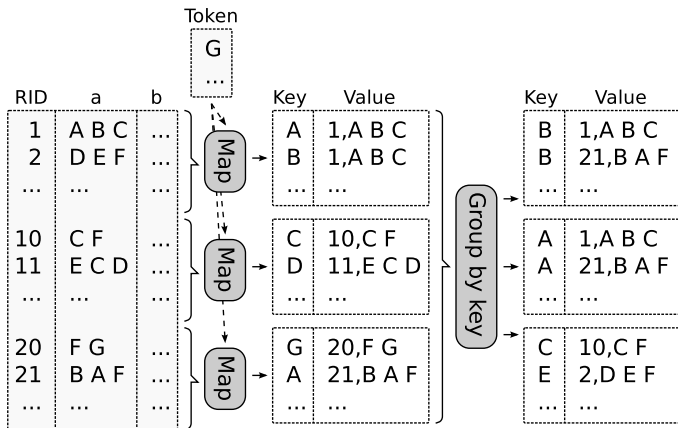
Stage 2: RID-Pair Generation



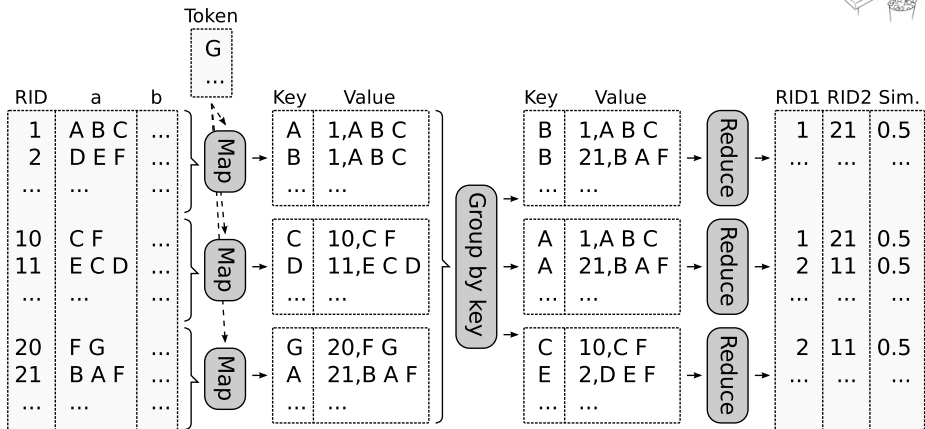
Stage 2: RID-Pair Generation



Stage 2: RID-Pair Generation



Stage 2: RID-Pair Generation



Additional Contributions



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



Experimental Setting

Hardware

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

Software

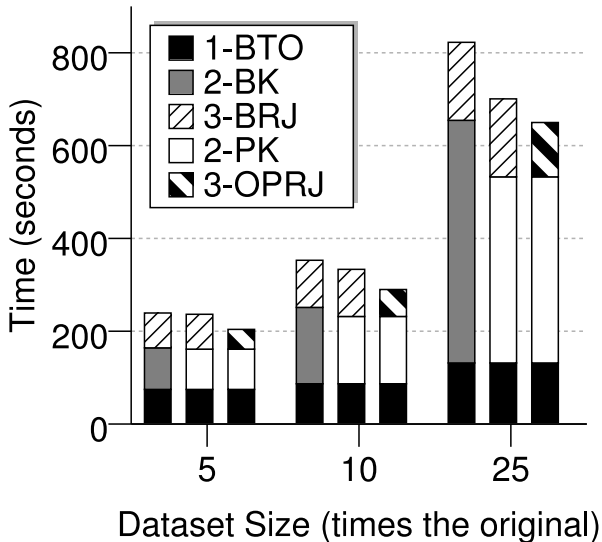
- Ubuntu 9.06, 64-bit, server edition OS
- Java 1.6, 64-bit, server
- Hadoop 0.20.1



Datasets

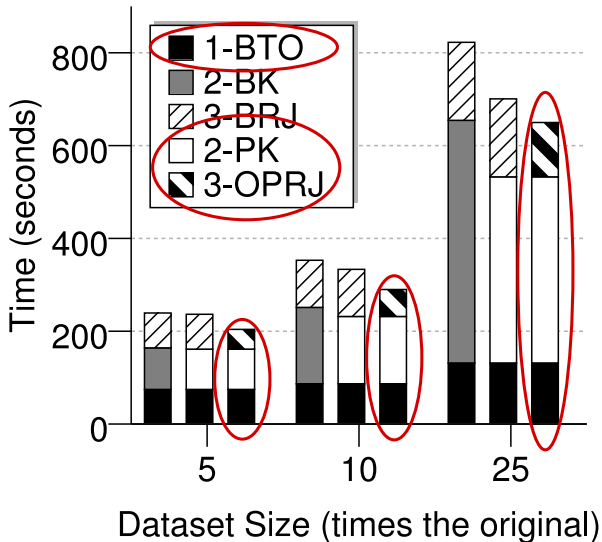
- DBLP
 - Average length: 259 bytes
 - Number of records: 1.2M
 - Total size: 300MB
- CITESEERX
 - Average length: 1374 bytes
 - Number of records: 1.3M
 - Total size: 1.8GB
- Increased each up to $\times 25$, preserving join properties
 - DBLP: 31M records, 8.2GB
 - CITESEERX: 32M records, 45GB

Running Time



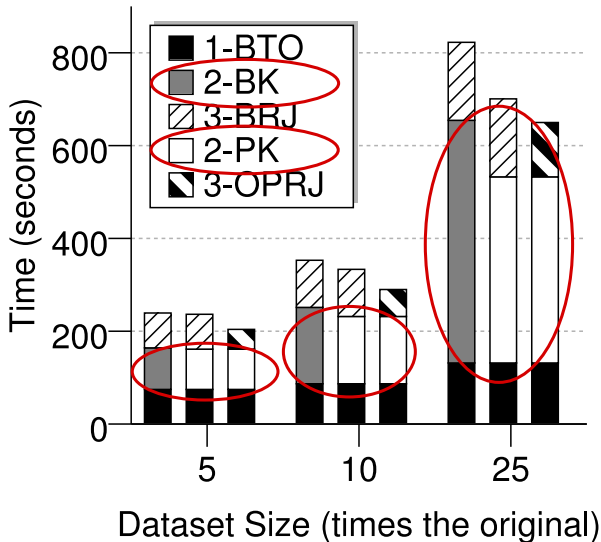
- Self-join $DBLP \times n$
- $n \in [5, 25]$
- 10-node cluster
- Best time
- Bulk of the time

Running Time



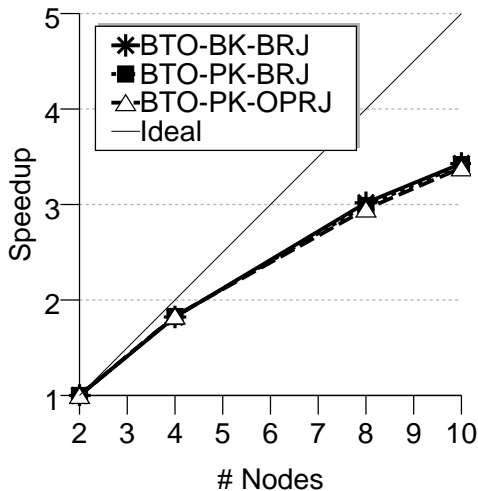
- Self-join DBLP $\times n$
- $n \in [5, 25]$
- 10-node cluster
- **Best time**
- Bulk of the time

Running Time

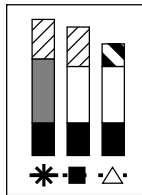


- Self-join $DBLP \times n$
- $n \in [5, 25]$
- 10-node cluster
- Best time
- **Bulk of the time**

Speedup

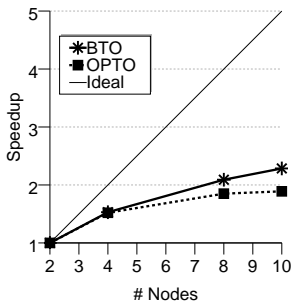


- Relative running time
- Self-join DBLP $\times 10$
- Different cluster sizes

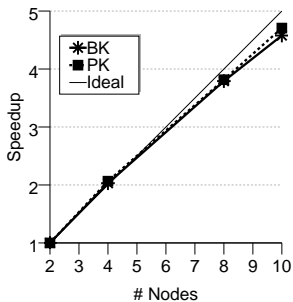


Speedup Breakdown

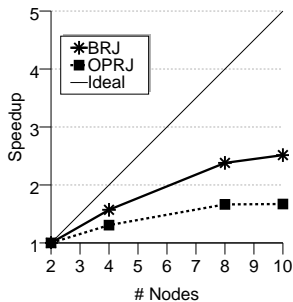
Stage 1



Stage 2

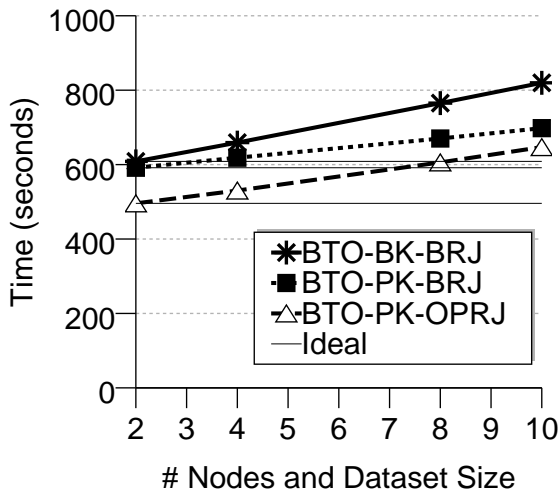


Stage 3

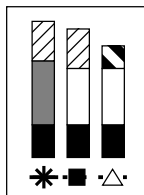


- Relative running time
- Self-join DBLP $\times 10$
- Different cluster sizes

Scaleup

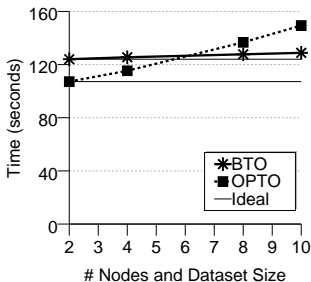


- Running time
- Self-joining DBLP $\times n$
- $n \in [5, 25]$
- Proportional cluster

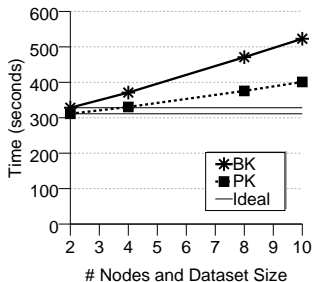


Scaleup Breakdown

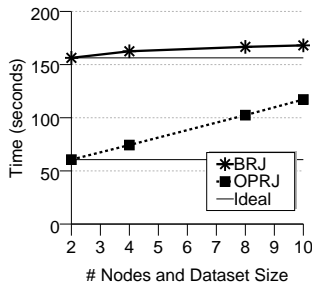
Stage 1



Stage 2



Stage 3



- 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
 - 40 cores, 40 disks cluster
 - Speedup and scaleup



<http://fog.ics.uci.edu>

A longer version of 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

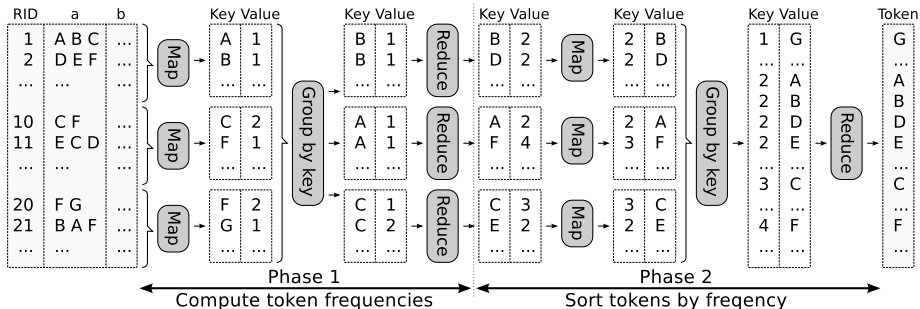
Flamingo

<http://flamingo.ics.uci.edu/>

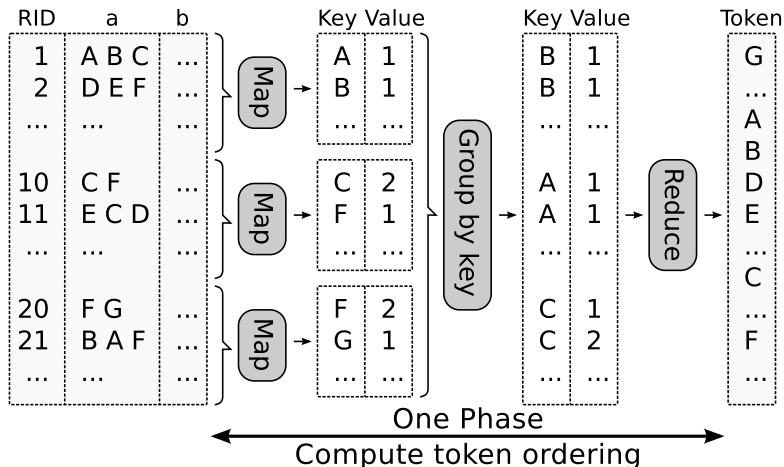
projects at UC Irvine.



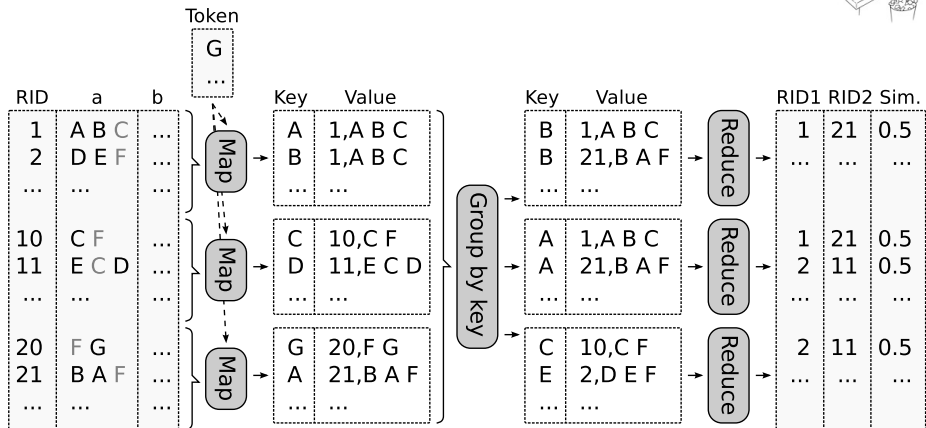
Stage 1: Basic Token Ordering (BTO)



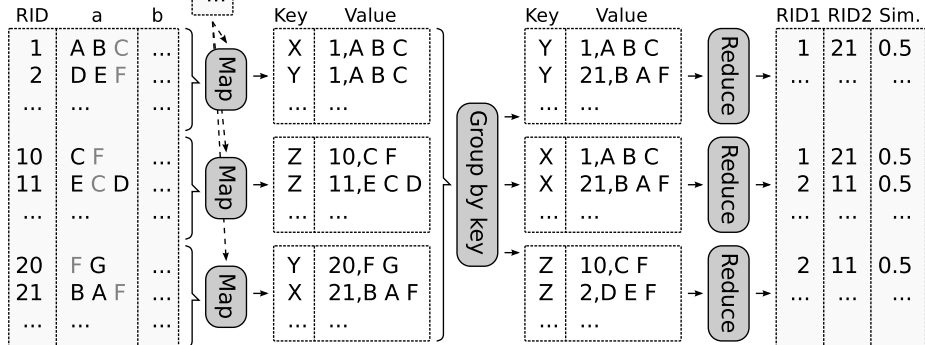
Stage 1: One Phase Token Ordering (OPTO)



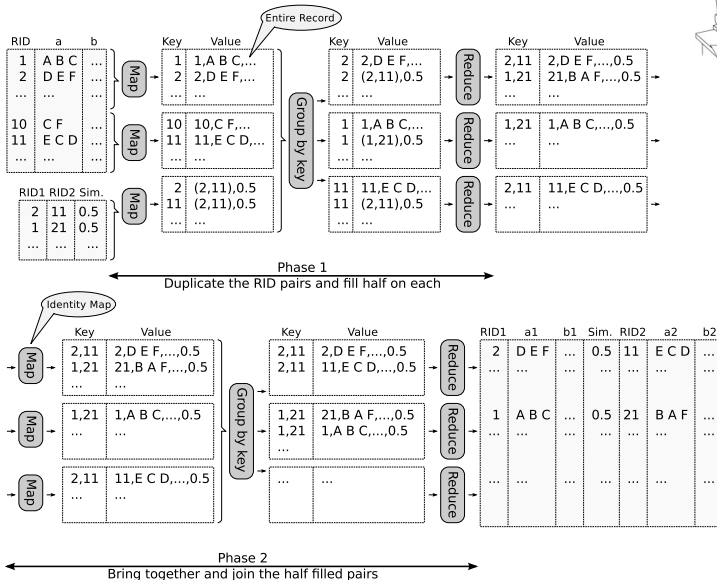
Stage 2: Basic Kernel (BK)



Stage 2: Indexed Kernel (PK)



Stage 3: Basic Record Join (BRJ)



Stage 3: One Phase Record Join (OPRJ)

