Fuzzy Joins Using MapReduce

Foto N. Afrati, et. al

April 15, 2013

Presented by Nathan Clement
Introduction

Motivation

Problem Definition
  \((M, C, R)\)-map-reducible algorithms
  Simplifying Assumptions

Algorithmic Details

Naïve

Ball-Hashing

Pigeon Hole (splitting)

Anchor Points

Results

Extensions

Applications to Other Distances

Conclusion
Motivation

**Take-Away**: There are many different correct and efficient algorithms that can be used in the MapReduce environment; however, none of them necessarily dominate any other. Instead, the correct choice of an algorithm is application-specific.

- Which properties should be adjustable? For this paper, only number of map-reduce steps maintained.
- Can be large Map phase, large Reduce phase, or high Communication
- Not running time, but theoretical algorithmic complexity
Problem Definition

- Given a dataset, $R$, with domain $\mathcal{D}$ and a similarity function $Sim : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$.
- Define a similarity function, also called a *Fuzzy-Join Predicate*, $F = (Sim, \tau)$ s.t.
  \[
  F(R) = \{(x, y) | x, y \in R, Sim(x, y) \geq \tau\}
  \]
- Return set of all pairs, $(x, y) \in F(R)$, that have similarity $> \tau$.
- Can be used in deduplication, searching, etc.
Problem Definition

- Given a dataset, $R$, with domain $\mathcal{D}$ and a similarity function $Sim : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$

- Define a similarity function, also called a Fuzzy-Join Predicate, $F = (Sim, \tau)$ s.t.

  $$F(R) = \{(x, y) | x, y \in R, Sim(x, y) \geq \tau\}$$

- Return set of all pairs, $(x, y) \in F(R)$, that have similarity $> \tau$.

- Can be used in deduplication, searching, etc.

- **Using MapReduce environment**
  
  - Input: Set, $S$, of input elements, similarity function $Sim$, and distance, $\tau$
  
  - Output: Pair $(s, t)$ s.t. $Sim(s, t) \geq \tau$
Define different MapReduce algorithms in terms of three variables:

- **M**: Total map or preprocessing cost across all input records
- **C**: Total communication cost of passing data from mappers to reducers
- **R**: Total computation cost of all reducers

\[ M + C + R \] is proportional to the “rent” the user has to pay for the resources.
Simplifying Assumptions

Some simplifying assumptions need to be made, but they should apply wlog.
Simplifying Assumptions

Some simplifying assumptions need to be made, but they should apply wlog.

- Number of *mappers* is never considered (can use as many as is necessary)
Simplifying Assumptions

Some simplifying assumptions need to be made, but they should apply wlog.

- Number of *mappers* is never considered (can use as many as is necessary)
- Unless explicitly stated, a *reducer* is just a single key and its associated value list, not a Reduce task on a compute node
Simplifying Assumptions

Some simplifying assumptions need to be made, but they should apply wlog.

- Number of mappers is never considered (can use as many as is necessary)
- Unless explicitly stated, a reducer is just a single key and its associated value list, not a Reduce task on a compute node
- Input set is random and unique, hash is optimal (distributes data equally)
Simplifying Assumptions

Some simplifying assumptions need to be made, but they should apply wlog.

- Number of mappers is never considered (can use as many as is necessary)
- Unless explicitly stated, a reducer is just a single key and its associated value list, not a Reduce task on a compute node
- Input set is random and unique, hash is optimal (distributes data equally)
- Operators perform at unit cost
  - copying (communication)
  - comparing
  - hashing
Simplifying Assumptions

Some simplifying assumptions need to be made, but they should apply wlog.

- Number of mappers is never considered (can use as many as is necessary)
- Unless explicitly stated, a reducer is just a single key and its associated value list, not a Reduce task on a compute node
- Input set is random and unique, hash is optimal (distributes data equally)
- Operators perform at unit cost
  - copying (communication)
  - comparing
  - hashing
- Costs are all “average”
Specific Algorithmic Details

Several algorithms were studied, only

- Describe for hamming distance, but other distances also can be used with slight modifications
  - hamming
  - edit
  - Jaccard

- Using hamming distance, so similarity function requires $d(x, y) < \tau$

- Ball-Hashing (2)
- Pigeon Hole (1)
- Anchor Points (2)
Naïve Algorithm

Have an arbitrary number, $K$, of reducers, set up in a triangle as follows:

1. Let each reducer be identified by a pair, $(i, j)$, s.t. $0 \leq i \leq j < J$ for some constant $J$. The number of reducers is $K = \binom{J+1}{2} = J(J + 1)/2$

2. Hash members of input set $S$ to $J$ buckets, sent to all reducers with either $(i, j)$ or $(j, i)$ (exactly $J$ reducers)

   - **Cost of Communication** is $O(|S| J) = O(|S\sqrt{K})$

3. Each reducer has $|S| J / K = 2|S| / (J + 1)$ elements, which requires $\left(\frac{2|S|}{(J + 1)}\right) = O(|S|^2 / K)$ comparisons.

   - For $K$ reducers, **Reducer Cost** is $O(|S|^2)$
Naïve Algorithm

Have an arbitrary number, $K$, of reducers, set up in a triangle as follows:

1. Let each reducer be identified by a pair, $(i, j)$, s.t. $0 \leq i \leq j < J$ for some constant $J$. The number of reducers is $K = \binom{J+1}{2} = J(J+1)/2$

2. Hash members of input set $S$ to $J$ buckets, sent to all reducers with either $(i, j)$ or $(j, i)$ (exactly $J$ reducers)
   - **Cost of Communication** is $O(|S|J) = O(|S\sqrt{K})$

3. Each reducer has $|S|J/K = 2|S|/(J+1)$ elements, which requires $\left(\frac{2|S|}{(J+1)}\right)^2 = O(|S|^2/K)$ comparisons.
   - For $K$ reducers, **Reducer Cost** is $O(|S|^2)$

- With $J \approx \sqrt{K}$, algorithm is $(M, C, R) = (|S|\sqrt{K}, |S|\sqrt{K}, |S|^2)$
Naïve Algorithm (continued)

Let $J$ be 4, then $K$ is 10, and reducer labels are the following:

- $(0,0)$
- $(1,1)$
- $(2,2)$
- $(3,3)$
- $(0,1)$
- $(1,2)$
- $(2,3)$
- $(0,2)$
- $(1,3)$
- $(0,3)$
Naïve Algorithm (continued)

- Let $J$ be 4, then $K$ is 10, and reducer labels are the following:
  - $(0,0)$  $(1,1)$  $(2,2)$  $(3,3)$
  - $(0,1)$  $(1,2)$  $(2,3)$
  - $(0,2)$  $(1,3)$
  - $(0,3)$
- Mapping cost is $J = 4$
- Communication cost is $J = 4 \approx \sqrt{K}$
- Reducing cost is $O(|S|^2)$
Naïve Algorithm (continued)

Let \( J \) be 4, then \( K \) is 10, and reducer labels are the following:
- \((0,0)\)
- \((1,1)\)
- \((2,2)\)
- \((3,3)\)
- \((0,1)\)
- \((1,2)\)
- \((2,3)\)
- \((0,2)\)
- \((1,3)\)
- \((0,3)\)

- Mapping cost is \( J = 4 \)
- Communication cost is \( J = 4 \approx \sqrt{K} \)
- Reducing cost is \( O(|S|^2) \)
- With \( J \approx \sqrt{K} \), algorithm is
  \((M, C, R) = (|S|\sqrt{K}, |S|\sqrt{K}, |S|^2)\)
Ball-Hashing: Introduction

Given a set, $S$, of $b$-bit strings and an integer $0 \leq d \leq b$, find the set:

$$\{(s_1, s_2) | HD(s_1, s_2) \leq d\}$$

$B(b, d)$ denotes the number of $b$-bit strings that can be obtained by flipping at most $d$ bits

$$B(b, d) = \sum_{k=1}^{d} \binom{b}{k} \approx \frac{b^d}{d!}$$

"ball of radius $d$"

When $b$ is clear from the context, just $B(d)$
Ball-Hashing I: unlimited reducers

- One reducer for each of the $n$ possible strings of length $b$.

1. Mappers send string to each $b$-bit string at most $d$ from it ($B(d)$)
   - Sends pair $(s, -1)$ and $(t, s)$ if $t \neq s$
2. If first element is $-1$, reducer emits $s$ and all $t$

Average number of strings sent to reducers is $|S|B(d)$
Number of strings received by reducers is $|S|B(d)/n$, total work is $|S|^2B(d)/n$

Algorithm is $(M, C, R) = (|S|B(d), |S|B(d), |S|^2B(d)/n)$ for $n$ reducers
Ball-Hashing I: unlimited reducers

- One reducer for each of the $n$ possible strings of length $b$.

1. Mappers send string to each $b$-bit string at most $d$ from it ($B(d)$)
   - Sends pair $(s, -1)$ and $(t, s)$ if $t \neq s$
2. If first element is $-1$, reducer emits $s$ and all $t$

- Average number of strings sent to reducers is $|S|B(d)$
- Number of strings received by reducers is $|S|B(d)/n$, total work is $|S|^2B(d)/n$
Ball-Hashing I: unlimited reducers

- One reducer for each of the $n$ possible strings of length $b$.

1. Mappers send string to each $b$-bit string at most $d$ from it ($B(d)$)
   - Sends pair $(s, -1)$ and $(t, s) | t \neq s$
2. If first element is $-1$, reducer emits $s$ and all $t$

- Average number of strings sent to reducers is $|S|B(d)$
- Number of strings received by reducers is $|S|B(d)/n$, total work is $|S|^2B(d)/n$
- Algorithm is $(M, C, R) = (|S|B(d), |S|B(d), |S|^2B(d)/n)$ for $n$ reducers
Ball-Hashing II: limited reducers

1. Still have \( n \) reducers, but only send \((s, t)\) to reducers distance \( d/2 \) or less.

2. Pairwise comparison of each \( s \) and \( t \)
   - Find all strings, \( U \), at most \( d/2 \) from both \( s \) and \( t \)

3. Each \( n \) reducers emits string \( u \) iff lexicographically first among all \( U \)
Algorithm to find lexicographically first:

- Given strings $s$ and $t$ that are distance $e \leq d$, and scanning $s$ from left to right:
  - change 1’s to 0’s where $s$ has 1 and $t$ has 0
  - change 1’s to 0’s where $s$ has 1 and $t$ has 1 if haven’t encountered $(d - e)/2$ positions with $s = 1$, $t = 1$
  - stop after changing $d/2$ 1’s to 0’s
- Example, let $d = 6$, and $(d - e)/2 = 1$:
  
  $s = 101101001100$
  $s' = 101101001100$
  $t = 101001101001$
Algorithm to find lexicographically first:

- Given strings $s$ and $t$ that are distance $e \leq d$, and scanning $s$ from left to right:
  - change 1’s to 0’s where $s$ has 1 and $t$ has 0
  - change 1’s to 0’s where $s$ has 1 and $t$ has 1 \textit{if haven’t encountered} $(d - e)/2$ \textit{positions with} $s = 1$, $t = 1$
  - stop after changing $d/2$ 1’s to 0’s

- Example, let $d = 6$, and $(d - e)/2 = 1$:
  - $s = 101101001100$
  - $s' = 001101001100$
  - $t = 101001101001$
Algorithm to find lexicographically first:

1. Given strings $s$ and $t$ that are distance $e \leq d$, and scanning $s$ from left to right:
   - change 1’s to 0’s where $s$ has 1 and $t$ has 0
   - change 1’s to 0’s where $s$ has 1 and $t$ has 1 if haven’t encountered $(d – e)/2$ positions with $s = 1, t = 1$
   - stop after changing $d/2$ 1’s to 0’s

2. Example, let $d = 6$, and $(d – e)/2 = 1$:
   - $s = 101101001100$
   - $s' = 001001001100$
   - $t = 101001101001$
Algorithm to find lexicographically first:

- Given strings $s$ and $t$ that are distance $e \leq d$, and scanning $s$ from left to right:
  - change 1’s to 0’s where $s$ has 1 and $t$ has 0
  - change 1’s to 0’s where $s$ has 1 and $t$ has 1 if haven’t encountered $(d - e)/2$ positions with $s = 1$, $t = 1$
  - stop after changing $d/2$ 1’s to 0’s

- Example, let $d = 6$, and $(d - e)/2 = 1$:
  - $s = 101101001100$
  - $s' = 001001001000$
  - $t = 101001101001$
Algorithm to find lexicographically first:

- Given strings $s$ and $t$ that are distance $e \leq d$, and scanning $s$ from left to right:
  - change 1's to 0's where $s$ has 1 and $t$ has 0
  - change 1's to 0's where $s$ has 1 and $t$ has 1 if haven’t encountered $(d - e)/2$ positions with $s = 1$, $t = 1$
  - stop after changing $d/2$ 1's to 0's
- Example, let $d = 6$, and $(d - e)/2 = 1$:
  - $s = 101101001100$
  - $s' = 001001001000$
  - $t = 101001101001$
- $d(s, s') = 3$, $d(t, s') = 3$
Ball-Hashing II: limited reducers

1. Still have $n$ reducers, but only send $(s, t)$ to reducers distance $d/2$ or less.
2. Pairwise comparison of each $s$ and $t$
   - Find all strings, $U$, at most $d/2$ from both $s$ and $t$
3. Each $n$ reducers emits string $u$ iff lexicographically first among all $U$
Ball-Hashing II: limited reducers

1. Still have $n$ reducers, but only send $(s, t)$ to reducers distance $d/2$ or less.
2. Pairwise comparison of each $s$ and $t$
   - Find all strings, $U$, at most $d/2$ from both $s$ and $t$
3. Each $n$ reducers emits string $u$ iff lexicographically first among all $U$

- Total cost:
  - Number of reducers: $n$
  - $M = C = |S|B(d/2)$
  - $R = |S|^2(B(d/2))^2b/n$
Pigeon Hole, or splitting

1. Split the $b$-bit strings into $d + 1$ equal-length substrings:
   $s_1 s_2 s_3 \cdots s_{d+1}$.
   - if $HD(s, t) \leq d$, at least one substring must be an exact match

2. Hash with key of $(i, s_i)$, value is $s$
   - $2^{b/(d+1)}$ possible substrings, total of $(d + 1)2^{b/(d+1)}$ reducers
   - Each reducer does a pairwise comparison to find the strings $s$ and $t$ that are at most distance $d$

3. Output $(s, t)$ if reducer received $s_i$ and there is no other $j < i$ for which $s$ and $t$ are equal
Pigeon Hole, or splitting

1. Split the $b$-bit strings into $d + 1$ equal-length substrings:
   $s_1s_2s_3 \cdots s_{d+1}$.
   ▶ if $HD(s, t) \leq d$, at least one substring must be an exact match

2. Hash with key of $(i, s_i)$, value is $s$
   ▶ $2^{b/(d+1)}$ possible substrings, total of $(d + 1)2^{b/(d+1)}$ reducers
   ▶ Each reducer does a pairwise comparison to find the strings $s$ and $t$ that are at most distance $d$

3. Output $(s, t)$ if reducer received $s_i$ and there is no other $j < i$ for which $s$ and $t$ are equal

▶ Total cost:
  ▶ Number of reducers: $(d + 1)n^{1/(d+1)}$
  ▶ $M = C = (d + 1)|S|$
  ▶ $R = (d + 1)|S|^2/n^{1/(d+1)}$
Anchor Points I: hamming codes

The Hamming Codes algorithm is a special version of the Cover Set algorithm where $d = 1$ and $b$ is one less than a power of 2.

- Several properties of a *Hamming code*:
  1. Number of strings is $n/b + 1$
  2. Every string is either in the Hamming code or at distance 1 from a unique member of the code.
  3. Can determine if $s$ is in the code (or if not, which member it is closest to) in time $O(b \log b)$.
The Hamming Codes algorithm is a special version of the Cover Set algorithm where $d = 1$ and $b$ is one less than a power of 2

- Several properties of a *Hamming code*:
  1. Number of strings is $n/b + 1$
  2. Every string is either in the Hamming code or at distance 1 from a *unique* member of the code
  3. Can determine if $s$ is in the code (or if not, which member it is closest to) in time $O(b \log b)$
1. Check if $s$ is in the code
   - TRUE: send $s$ to reducer for $s$
   - FALSE: send $s$ to reducer for $t$ (string corresponding to code at distance 1)

2. For all strings that are distance 1 from $s$ (flip bits), send $s$ to codeword at distance 1
   - Work by mappers for each input string is $O(b^2 \log b)$, only communication of $b$

3. Builds index for received words, $U$, outputs:
   3.1 $s$ and all received strings, $t \in U$ if $s$ was received
   3.2 all $u \in U$ s.t. $d(t \in U, u) = 1$ and $u$ lexicographically before $t$ (avoids duplicates)
Anchor Points I: hamming codes (continued)

- Assume each reducer receives average number of strings,
  - receives $b|S|/n$ strings at distance 1 (step 1)
  - receives $\binom{b}{2}|S|/n$ strings at distance 2 (step 2)
- Index built allows $O(1)$ time, so $O(b)$ time to find all strings $d = 1$ (step 3.1)
- For each $b|S|/n$, flips $b - 1$ bits and tests for membership (step 3.2)
- Total Cost:
  - #reducers: $n/(b + 1)$
  - $M = b^2 \log b|S|$
  - $C = b|S|$
  - $R = \frac{b^2}{b+1}|S|$
Anchor Points II: cover set

Generalization of Hamming codes algorithm

- Set of $A$ anchor points, indexed at each mapper
  - Best performance requires *perfect code*, of which there exist almost none.
  - Can *almost* get $n/B(d)$
Anchor Points II: cover set (continued)

1. Send \( s \) to every anchor point at distance at most \( 2d \) from \( s \)
2. Depending on \( |A| \), \( b \), and \( d \):
   2.1 Generate all \( t \), test for membership in \( A \) (work \( O(B(2d)) \))
   2.2 Consider all anchor points and test distance from \( s \) (work \( O(n/B(d)) \), but distance function?)
3. Tag each string with the nearest “home” reducer so it only outputs once
   - Each pair must have a home string, and home string must lexicographically proceed non-home string
   - Total Cost:
     - reducers: \( n/B(d) \)
     - \( M = |S| \left( \min(B(2d), \frac{n}{B(d)}) \right) \)
     - \( C = |S|B(2d)/B(d) \)
     - \( R = |S|B(d)/n \)
### Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Map-cost Per Element</th>
<th>Reducers</th>
<th>Communication (C)</th>
<th>Processing (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>$J$ (approx $\sqrt{K}$)</td>
<td>$K$ (arbitrary)</td>
<td>$</td>
<td>S</td>
</tr>
<tr>
<td>BH unlimited</td>
<td>$B(d)$</td>
<td>$n$</td>
<td>$</td>
<td>S</td>
</tr>
<tr>
<td>BH limited</td>
<td>$B(d/2)$</td>
<td>$n$</td>
<td>$</td>
<td>S</td>
</tr>
<tr>
<td>Pigeon Hole</td>
<td>$d + 1$</td>
<td>$(d + 1)n^{\frac{1}{d+1}}$</td>
<td>$(d + 1)</td>
<td>S</td>
</tr>
<tr>
<td>AP hamming</td>
<td>$b^2 \log b$</td>
<td>$n/(b + 1)$</td>
<td>$b</td>
<td>S</td>
</tr>
<tr>
<td>AP cover</td>
<td>$\min B(2d), \frac{n}{B(d)}$</td>
<td>$\frac{n}{B(d)}$</td>
<td>$</td>
<td>S</td>
</tr>
</tbody>
</table>
Results – specific

Let \( b = 20, d = 4, |S| = 10^5 \), and \( K = 10^4 \), then the following results hold:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Map-cost Per Element</th>
<th>Reducers</th>
<th>Communication (C)</th>
<th>Processing (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>100</td>
<td>10^4</td>
<td>10^7</td>
<td>10^{10}</td>
</tr>
<tr>
<td>BH unlimited</td>
<td>6226</td>
<td>10^5</td>
<td>6.2 \times 10^7</td>
<td>6.2 \times 10^8</td>
</tr>
<tr>
<td>BH limited</td>
<td>211</td>
<td>10^5</td>
<td>2.1 \times 10^7</td>
<td>4.5 \times 10^8</td>
</tr>
<tr>
<td>Pigeon Hole</td>
<td>5</td>
<td>80</td>
<td>5 \times 10^5</td>
<td>6.3 \times 10^8</td>
</tr>
<tr>
<td>AP hamming</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
</tr>
<tr>
<td>AP cover</td>
<td>160</td>
<td>160</td>
<td>4.2 \times 10^6</td>
<td>6.2 \times 10^8</td>
</tr>
</tbody>
</table>

- No algorithm necessarily dominates any other
- **Communication Cost**: Pigeon Hole < AP cover < Naïve < BH limited ≤ BH unlimited
- **Reducer Cost** (Processing): BH limited < BH unlimited ≤ AP cover < Pigeon Hole < Naïve
- While Naïve is dominated by Pigeon and AP, Naïve can adjust reducers, so with \( K = 1 \), communication is only \( 10^5 \)
Other Distance Metrics: Hamming Distance

- Claim the following property:
  Between strings $s_1$ and $s_2$ with lengths $b_1$ and $b_2$ and $l$ being the length of the longest common subsequence, the edit distance is:

  $$e(s_1, s_2) = (b_1 + b_2 - 2l)$$

- Two Sequences, $l = 4$
  1001101
  1011001

- Edit distance:
  $7 + 7 - 2*4 = 14 - 8 = 6$
Other Distance Metrics: Hamming Distance

- Claim the following property:
  
  Between strings $s_1$ and $s_2$ with lengths $b_1$ and $b_2$ and $l$ being the length of the longest common subsequence, the edit distance is:
  
  $$e(s_1, s_2) = (b_1 + b_2 - 2l)$$

- Two Sequences, $l = 4$
  
  1001101
  
  1011001

- Edit distance:
  
  $7 + 7 - 2 \times 4 = 14 - 8 = 6$

- Edits made:
  
  1. 1001101
  2. 101101 (one deletion)
  3. 1011001 (one insertion)
Other Distance Metrics: Jaccard Similarity

- **Jaccard similarity:**
  \[ J_{1,2} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \]

- **Jaccard distance between sets:**
  \[ d_J = 1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \]

- "essentially the edit-distance algorithms applied to sorted string representation of sets that comes from Chaudhuri et al..."
Conclusion

- Similarity joins in a *single* map-reduce step
- Compared based on
  1. map cost
  2. reducer cost
  3. communication cost
- Multiple non-dominanted algorithms, choose depending on application

Lingering question: Is there one that dominates the other by total work done? Is there a notion of "wasted" effort?
Conclusion

- Similarity joins in a *single* map-reduce step
- Compared based on
  1. map cost
  2. reducer cost
  3. communication cost
- Multiple non-dominated algorithms, choose depending on application
- Lingering question: Is there one that dominates the other by *total* work done? Is there a notion of “wasted” effort?