Distributed Data Deduplication

Xu Chu, Ihab F. Ilyas, Paraschos Koutris

PVLDB 9(11), 2016

Presenter: David Liu

20428295

Data deduplication is costly

- **Data Deduplication is** the process of identifying tuples that refer to the same entity
- Need to compute similarity measures of each tuple pair – $O(n^2)$ complexity!

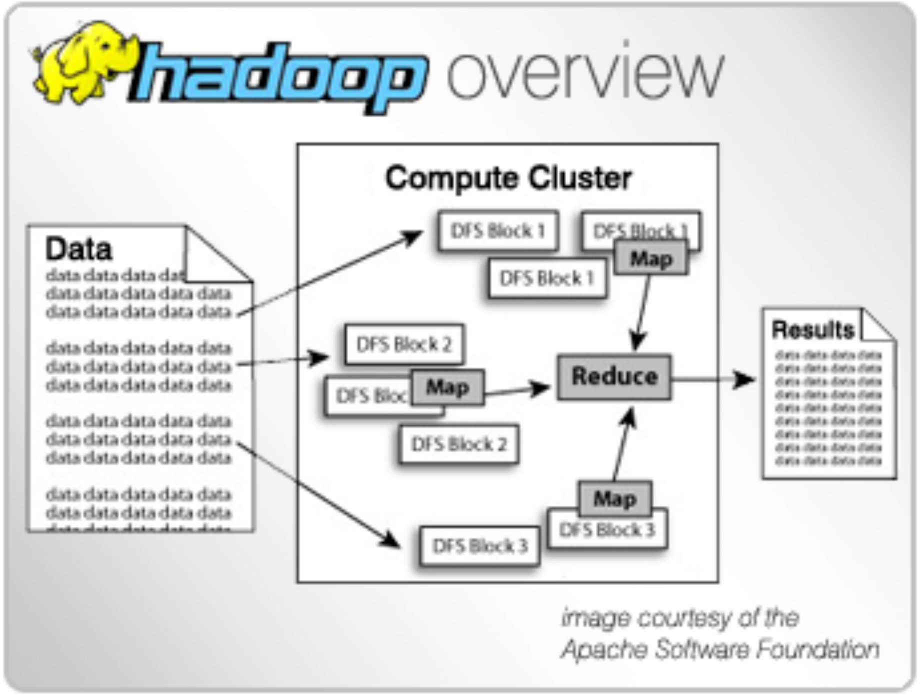
```
DELETE FROM
(
  SELECT
    *,
    ROW_NUMBER() OVER (PARTITION BY acct_id
                       ORDER BY acct_open_dt) AS row_num
  FROM accounts
)
WHERE row_num > 1;
```

Blocking is often used to reduce workload

- **Blocking functions** divide a relation into subsets of tuples
    - Tuples across subsets are considered dissimilar
- E.g., edit distance, Jaccard similarity, postal code
- Requires familiarity with the data and hand tuning
- Can also be adaptive (cluster learning)
- Multiple blocking functions are often used to reduce false negatives

Distributed/parallel data processing is faster than running on a single machine

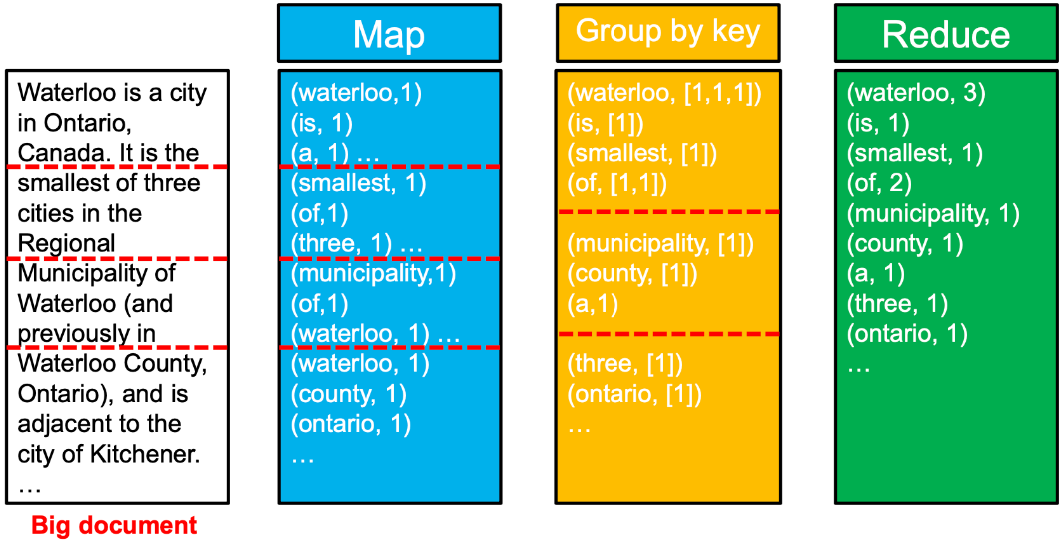
- Multiple workers in parallel
  - Shared-nothing structure - each worker has own memory and disk
  - Trade-off between **communication cost** and **computation cost**
- Challenges
  - Network transfer time, disk I/O time
  - Load-balancing
  - Need to handle multiple blocking functions



Hadoop MapReduce

- Mapper
  - Takes in raw data
  - Does parsing/transformation/filtering
  - Outputs (key, value) pair
- Partitioner
  - Sorts and shuffles
  - Outputs (key, iterable(value)) pair
  - Pairs with the same key gets assigned to the same reducer
- Reducer
  - Does aggregation (in our case, compare)
  - Outputs results

MapReduce "word count" example



Source: University of Waterloo, CS651, Fall 2019

Contributions

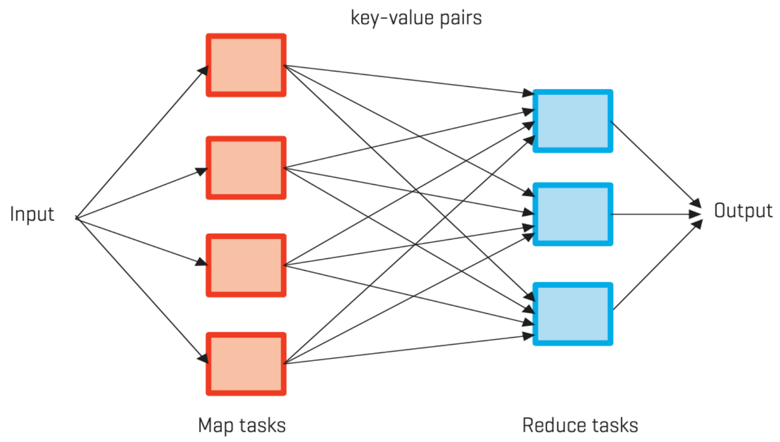
- Formalizes a computation/memory cost model
- Proposes an optimal workload distribution strategy
    - Small constant factor from the theoretical lower bounds
    - Balance between computational cost and communication cost
    - Handles both single and multiple blocking functions
- Does NOT propose new comparison methods
- Does NOT propose new blocking functions

Problem Definition

Communication cost (X) and computation cost (Y)

- **Communication cost** is the cost of moving the data from Map tasks to Reduce tasks
- **Computation cost** is the sum of the time needed to execute each reducer

Figure 1: The structure of a MapReduce job.



Source: J. D. Ullman., Designing Good MapReduce Algorithms

$$n = \text{ number of tuple}$$

$$h_i, i \in \{1, ..., s\} = \text{ blocking functions}$$

$$B_j, j \in \{1, ..., m\} = \text{ generated blocks}$$

$$k = \text{ number of workers}$$

$$X_i = \text{ communication cost for worker } i$$

$$Y_i = \text{ computation cost for worker } i$$

$$X = \max_{i \in [1,k]} X_i$$

$$Y = \max_{i \in [1,k]} Y_i$$

Example:
n = 100, k = 10, s = 1, blocks = {5 blocks of size 10, 25 blocks of size 2}

- Strategy #1
  - Send all 100 tuples to every reducers
  - Proportionally distribute tuples to compare among the reducers

$$X_i = 100$$

$$Y_i = \frac{5 \cdot \binom{10}{2} + 25 \cdot \binom{2}{2} = 250}{10} = 25$$

$$\Rightarrow X = 100, Y = 25$$

- Strategy #2
  - Assigns one block entirely to one reducer
  - 5 blocks of size 10 – 5 workers, 1 each
  - 25 blocks of size 2 – 5 workers, 5 each

$$X_i = 1 \cdot 10 = 5 \cdot 2 = 10$$

$$Y_i = \begin{cases} 1 \cdot \binom{10}{2} = 45, i \in \{1, ..., 5\} \\ 5 \cdot \binom{2}{2} = 5, i \in \{6, ..., 10\} \end{cases}$$

$$\Rightarrow X = 10, Y = 45$$

Break down into three cases, progressive in complexity

1. **Single** block generated by a **single** blocking function
2. **Multiple** blocks generated by a **single** blocking function
3. **Multiple** blocks generated by **multiple** blocking functions

Case 1 – Single Block, Single Blocking Function

Compare every tuple with every other tuple in the block

- It can be shown that, for a
  - Single block of size of n
  - k reducers
- The **lower bounds** on the **maximum costs** (i.e., X and Y) are

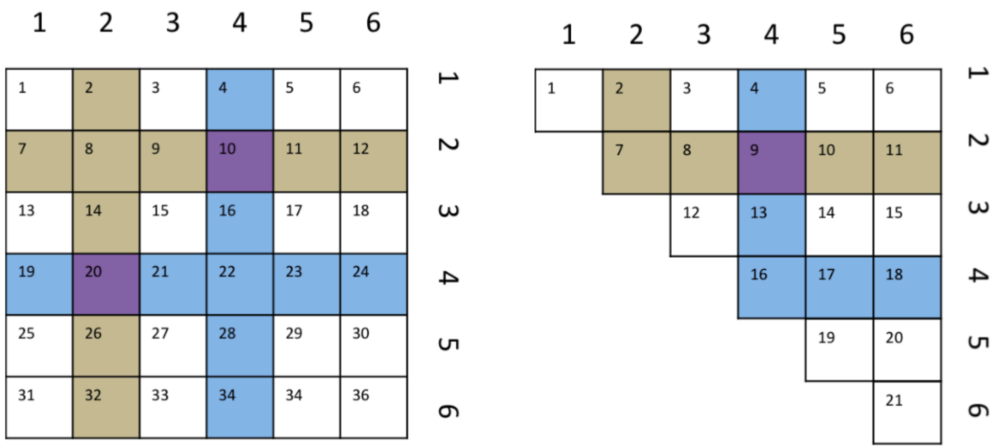
$$X_{low} = \frac{n}{\sqrt{k}}$$

$$Y_{low} = \frac{n(n-1)}{2k}$$

- We want to design a distribution strategy that gets as close to these lower bounds as possible

Triangle distribution strategy ensures exactly once comparison

- **Anchor points** are used to randomly assign tuple to reducers
- **Flags** (L, R, or S) are used to ensure exactly once comparison in reducer
    - L vs. R
    - S vs. S



(a) $R \times S$ join (b) Self-join

Figure 1: Reducer arrangement. (The number in the upper left corner of each cell is the reducer id.)

Example

Tuple	Anchor
A	2
B	4
C	4

Mapper Outputs		
(2, L#A)	(4, L#B)	(4, L#C)
(7, S#A)	(9, L#B)	(9, L#C)
(8, R#A)	(13, L#B)	(13, L#C)
(9, R#A)	(16, S#B)	(16, S#C)
(10, R#A)	(17, R#B)	(17, R#C)
(11, R#A)	(18, R#B)	(18, R#C)

Partition Outputs	Reducer Comparisons
(2, {L#A})	
(4, {L#B, L#C})	
(7, {S#A})	
(8, {R#A})	
(9, {R#A, L#B, L#C})	A v.s. B, A v.s. C
(10, {R#A})	
(11, {R#A})	
(13, {L#B, L#C})	
(16, {S#B, S#C})	B v.s. C
(17, {R#B, R#C})	
(18, {R#B, R#C})	

1	2	3	4	5	6	q/p
1	2	3	4	5	6	**1**
	7	8	9	10	11	**2**
		12	13	14	15	**3**
			16	17	18	**4**
				19	20	**5**
					21	**6**

Caveat?

Triangle distribution strategy matches lower bound on Y, while providing a constant-factor approximation to lower bound on X

$$X \leq (1 + o(1))\sqrt{2}X_{low} \approx \frac{\sqrt{2}n}{\sqrt{k}}$$

$$Y \leq (1 + o(1))Y_{low} \approx \frac{n(n-1)}{2k}$$

Case 2 – Multiple Blocks, Single Blocking Function

Like the single block case, we can derive lower bounds on X and Y

$\{B_1, ..., B_m\},$ blocks produced by blocking function h

$\{k_1, ..., k_m\},$ number of workers assigned to each block

$W_i = \binom{|B_i|}{2},$ number of comparisons in block i

$W = \sum_{i=1}^{m} W_i,$ total number of comparisons

$$X_{low} = \max\left(\frac{n}{k}, \frac{\sqrt{2W}}{\sqrt{k}}\right)$$

$$Y_{low} = \frac{W}{k}$$

Baseline strategies – *Naïve-Dedup* and *PJ-Dedup*

$$X_{low} = \max\left(\frac{n}{k}, \frac{\sqrt{2W}}{\sqrt{k}}\right)$$

$$Y_{low} = \frac{W}{k}$$

- *Naïve-Dedup*
  - Assigns every block entirely to one reducer
  - Worst case scenario – one dominating block with all n tuples
    - Since it was sent entirely to one reducer, $X = n$ and $Y = \binom{n}{2} = W$
    - Both are k times worse than their lower bounds - bad!
- *PJ-Dedup*
  - Assigns all blocks to all reducers, and use *triangle distribution strategy*
    - Modify the Mapper function's output key to composite (rid, bkv)
  - $X \approx \frac{\sqrt{2}n}{\sqrt{k}}$ and $Y \approx \frac{W}{k}$, just like under single block

So … PJ-Dedup?

Costs vary drastically depending on the blocking function

- h_1
  - Small and uniform block sizes
  - *Naïve-Dedup* is optimal
  - Should use one reducer per block
- h_2
  - Dominating block
  - *PJ-Dedup* is optimal
  - Should send to all reducers to divide up the workload
- h_3
  - General - a few relatively large blocks, rests are small
  - *Naïve-Dedup* is not optimal; *PJ-Dedup* is optimal for Y only
  - **Should use multiple (not all) reducers for the large blocks**

	h_1		h_2		h_3	
	X	Y	X	Y	X	Y
Lower bounds	$\frac{n}{k}$	$\frac{W}{k}$	$\frac{n}{\sqrt{k}}$	$\frac{W}{k}$	$\frac{\sqrt{\beta}n}{k}$	$\frac{W}{k}$
Naive-Dedup	$\frac{n}{k}$	$\frac{W}{k}$	n	W	$\frac{\beta n}{k}$	$\frac{\beta W}{k}$
PJ-Dedup	$\frac{\sqrt{2}n}{\sqrt{k}}$	$\frac{W}{k}$	$\frac{\sqrt{2}n}{\sqrt{k}}$	$\frac{W}{k}$	$\frac{\sqrt{2}n}{\sqrt{k}}$	$\frac{W}{k}$

Dis-Dedup: allocate reducers to blocks in proportion to workload

- B_i will be assigned to $k_i = \max(\lfloor \frac{W_i}{W} k \rfloor, 1)$ reducers
  - For those with $k_i = 1$, we call these *single-reducer blocks*
  - For those with $k_i > 1$, we call these *multi-reducer blocks*
- Every reducer handles **at most one** multi-reducer block and **at most one** single-reducer block
- Determining block type can be done beforehand using three WordCount-like MapReduce jobs
- Let
  - X_s/ X_l denote the maximum number of tuples from single-reducer blocks/multiple-reducer blocks received by any reducer
  - Y_s/ Y_l denote the maximum number of comparison from single-reducer blocks/multiple-reducer blocks performed by any reducer
- $X = X_s + X_l, Y = Y_s + Y_l$

Dis-Dedup can achieve constant factor to the lower bounds, for both multiple and single reducer blocks

- For multi-reducer blocks, *Dis-Dedup* uses the *triangle distribution strategy*

- For single-reducer blocks, *Dis-Dedup* uses a *hybrid distribution approach* to determine which reducer each block goes to
    - Some are deterministic, some are randomized

$$X_l \leq (1 + o(1))2X_{low}$$
$$Y_l \leq (1 + o(1))2Y_{low}$$

$$X_s \leq (1 + o(1))3X_{low}$$
$$Y_s \leq (1 + o(1))3Y_{low}$$

$$X = X_s + X_l \leq (1 + o(1))5X_{low}$$
$$Y = Y_s + Y_l \leq (1 + o(1))5Y_{low}$$

Case 3 – Multiple Blocks, Multiple Blocking Function

We cannot just run *Dis-Dedup* multiple times for multiple blocking functions

- There are now s blocking functions, $\{h_1, h_2, \ldots, h_s\}$
- Why not run *Dis-Dedup* s times?
  - Wasted communication cost by sending more tuples than necessary
  - Wasted computation costs by comparing same tuple pair more than once
- Solutions
  1. Allocate reducers to blocking functions in proportion to their workload
  2. Impose an ordering of the blocking functions

Dis-Dedup+: allocate reducers to blocks in proportion to workload **of that blocking function**

- A modified version of *Dis-Dedup*
- Instead of $k_i = \max\left(\lfloor \frac{w_i}{W} k \rfloor, 1\right)$, we now have $k_i^j = \max\left(\lfloor \frac{w_i^j}{W} k \rfloor, 1\right)$
  - The superscript j denotes the blocking function h_j
  - W is now the total workload generated by **all blocking functions**
- We can view this as a set of (possibly overlapping) blocks produced by one blocking function, and apply the normal *Dis-Dedup*
- Number of comparisons (Y) will be optimal, but the input (X) may have to be replicated

$$X \le (5s + o(1))X_{low}$$

$$Y \le (5 + o(1))Y_{low}$$

Dis-Dedup+: impose an ordering of blocking functions to avoid duplicate checking

- Order all s blocking functions, and send this order to all reducers
- Before a reducer compares two tuples t_1 and t_2 generated by h_j, it checks if any lower numbered blocking functions (i.e., $\{h_1, \dots, h_{j-1}\}$) also puts t_1 and t_2 in the same block
    - If it is the case, then there must be another reducer who is also doing that comparison

Experimental Studies

Contest participants

- *Dis-Dedup/Naïve-Dedup/PJ-Dedup*
- *Dedoop*
  - State-of-the-art
  - Equally distribute all tuple pairs among all reducers
  - Problem: requires much more memory
  - Only optimizes Y (computation cost), ignores X (communication cost)
- All the above, extended version (i.e., handles multiple blocking functions)

Experiments

	Description	Winner in X	Winner in Y	Winner in Time
1	Single block: varying number of tuples	Dis-Dedup	Dedoop	Dis-Dedup
2	Single block: Varying number of reducers	Dis-Dedup	Dedoop	Dis-Dedup
3	Multiple blocks, single blocking function: Varying number of blocks	Dis-Dedup	Dis-Dedup	Dis-Dedup
4	Multiple blocks, single blocking function: Varying block size distribution	Dis-Dedup	Dedoop	Dis-Dedup
5	Multiple blocks, single blocking function: Varying number of reducers	Dis-Dedup	Dis-Dedup	Dis-Dedup
6	Multiple blocks, multiple blocking functions: Varying number of block functions	Dis-Dedup+	Dis-Dedup+	Dis-Dedup+

Closing Remarks

Key Takeaways

- Both the communication and the computation costs are important when designing a distributed system
- Avoid duplicate comparisons when possible
- Strive for even load balancing
- Reduce memory footprint

Evaluation

- Accept
- Novel
- High technical details
- Adequate presentations

Future Work

- Reduce overlapping tuples sent under multiple blocking functions
- Cross-reducer inference?
  - If A == B and A == C, then don’t need to check if B == C
- Spark?
  - Less I/O time
  - Lazy execution

Thank you