Similarity Join

Younghoon Kim
(nongaussian@hanyang.ac.kr)

Similarity Operations

- Similarity search vs. join vs. self-join
- Usually, the heaviest step in many data mining techniques e.g., clustering
- Similar objects?
  - Distance (similarity) measure
  - Threshold

Similarity Join

- Find all similar pairs of cakes
  - Similar cakes: A pair of cakes whose distance is at most the given threshold

Threshold = 0.8

Two difficulties
(1) heavy computation cost for evaluating the similarity between cakes
(2) too much computation time with large data

Similarity Join: Problem Definition

- Given:
  - High dimensional data points $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots$
    - For example: Image is a long vector of pixel colors
  - A distance function $\boldsymbol{d}(\boldsymbol{x_1}, \boldsymbol{x_2})$ which quantifies the "distance" between $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$
  - A distance threshold $\boldsymbol{s}$
- Goal:
  - Find *all pairs of data points* $(\boldsymbol{x_i}, \boldsymbol{x_j})$ that are within some distance threshold $\boldsymbol{d}(\boldsymbol{x_i}, \boldsymbol{x_j}) \leq \boldsymbol{s}$
- Note:
  - Naïve solution would take $\boldsymbol{O}(\boldsymbol{N}^2)$
  - where $\boldsymbol{N}$ is the number of data points

Theta Joins

- Use primitive comparison operators (<,>,≤,≥,≠,=) in the join-predicates

```
SELECT *
FROM R, S
WHERE R.a > S.a;
```

R

	r_{id}	a
r_1	1	1
r_2	2	1
r_3	3	2
r_4	4	3

S

	s_{id}	a
s_1	1	1
s_2	2	1
s_3	3	2
s_4	4	2
s_5	5	3
s_6	6	4

Similarity Joins

- Given
  - A distance measure d
  - A threshold σ

```
SELECT *
FROM R, S
WHERE d((R.1, R.2, R.3),(S.1, S.2, S.3))
≤ σ
```

R

id	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

S

id	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

Similarity Self-Joins

- Given
  - A distance measure d
  - A threshold σ

```
SELECT *
FROM D as R, D as S
WHERE
R.id < S.id and
d((R.1, R.2, R.3),(S.1, S.2, S.3)) ≤ σ
```

D

id	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

1. Jaccard similarity
2. Euclidean distance
3. Cosine distance
4. Hamming distance

DISTANCE MEASURES

Distance Metric

- A distance metric on this space is a function $d(x, y)$ that takes two points in the space as arguments and produces a real number, and satisfies the following axioms:
  - **Non-negativit**y: $d(x, y) \geq 0$
  - **Reflexivity**: $d(x, y) = 0$ if and only if $x = y$ (distances are positive, except for the distance from a point to itself)
  - **Symmetry**: $d(x, y) = d(y, x)$
  - **Triangular inequality**: $d(x, y) \leq d(x, z) + d(z, y)$

Euclidean Distance

- An n-dimensional Euclidean space
  - points are vectors of n real numbers
- The conventional distance measure in this space,
  - which we shall refer to as the L_2-norm, is defined:

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Euclidean Distance

- L_r-distance

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r}$$

- L_1-distance
  - Manhattan distance
- L_∞-distance
  - the maximum of $|x_i - y_i|$ over all dimensions i

Exercise

- Question 2.:
  - Consider the two-dimensional Euclidean space (the customary plane) and the points (2, 7) and (6, 4).
  - What is Euclidean distance?
  - What is L_1-norm?
  - What is L_∞-norm?

Jaccard Similarity

- Jaccard coefficient/similarity
  - The Jaccard similarity of two sets is the size of their intersection divided by the size of their union:

$$\mathbf{sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|}$$

- Jaccard distance: $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$

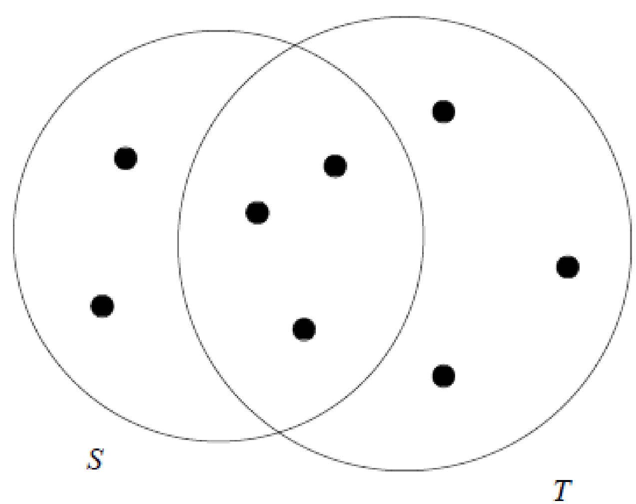


Figure 3.1: Two sets with Jaccard similarity 3/8

Exercise

- Compute the Jaccard similarities of each pair of the following ***three sets***:
- {1, 2, 3, 4}, {2, 3, 5, 7}, and {2, 4, 6}.

Exercise

- Prove that Jaccard distance is also a metric, satisfying the triangular inequality
  - $d(x_1, x_2) = 1 - \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|}$

Cosine Distance

- Given
  - two vectors x and y,
- The cosine of the angle between them is
  - the dot product x·y divided by the L_2-norms of x and y (i.e., their Euclidean distances from the origin).

$$\cos(x, y) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^{\,2}}\sqrt{\sum_{i=1}^{d} y_i^{\,2}}}$$

Exercise

- Question 3.:
  - Let
    - our two vectors be x = [1, 2,−1] and y = [2, 1, 1]
  - Cosine of the angle between x and y?

Exercise

- Prove that cosine distance is also a metric, satisfying the triangular inequality
  - $d(x_1, x_2) = 1 - \frac{x_1 \cdot x_2}{|x_1| \cdot |x_2|}$

MAPREDUCE PROGRAMMING TO PROCESS SIMILARITY JOINS

Basic Algorithms to Process Similarity Joins

- Nested loop join
  - Compute the distances of all possible pair of objects
  - Time complexity: $O(n^2)$
- Block-nested loop join
  - Compute all distances too ➔ Time complexity: $O(n^2)$
  - But it considers the memory hierarchy that big data is on disks while data for computations should be on the main memory

Nested Loop Self-Join

- **for each** tuple t_i **in** r **do begin**

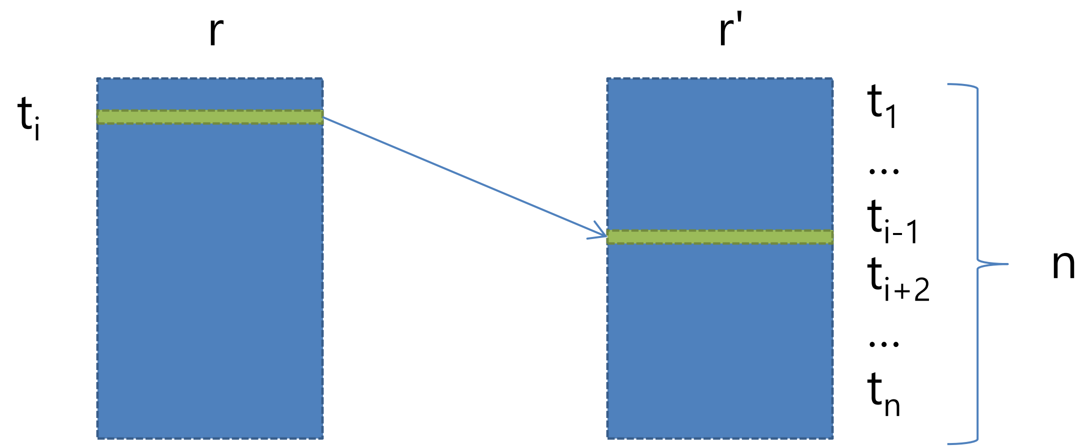
  for each tuple t_j **located next to** $\underline{t_r}$ **in** r **do begin**

  test pair (t_i, t_j) to see if they satisfy the join condition θ

  if they do, add $t_i \bullet t_j$ to the result.

  end

  end



Thinking in MapReduce

1. Define mapper’s input.
   - “i, <d-dimensional data points, p_i>”
2. Define reducer’s key.
   - (i, j) → the reducer computes $d(p_i, p_j)$
3. Define the key-value pairs in mapper.
   - $<(1, i), p_i>, <(2, i), p_i>, \ldots, <(i-1, i), p_i>, <(i, i+1), p_i>, \ldots, <(i, n), p_i>$
4. Define the output in reducer.
   - Compute $d(p_i, p_j)$

Nested Loop Join-like MapReduce?

With a data points t_i,

$< (1, i), t_i >$

$< (2, i), t_i >$

…

$< (i-1, i), t_i >$

$< (i, i+1), t_i >$

…

$< (i, n), t_i >$

groupByKey

$< (i, j), (t_i, t_j) >$

filter

$< (i, j), d(t_i, t_j) >$

PySpark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
  .master("local[*]")\
  .getOrCreate()
sc = spark.sparkContext
```

```
n = 2000
B = 10
```

```
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=n, centers=10, n_features=32,
                  random_state=0)
```

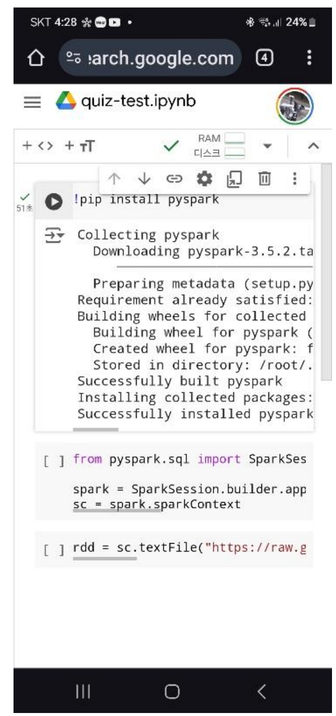
```
rdd = sc.parallelize([ (i, x) for i, x in enumerate(X) ])
```

PySpark

- Complete the PySpark code to run nested loop join-like MapReduce process

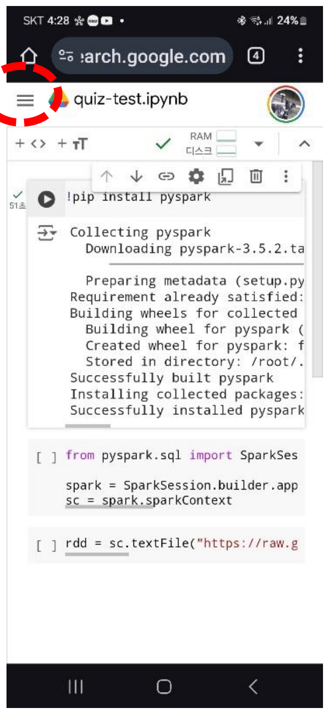
Copy Template File

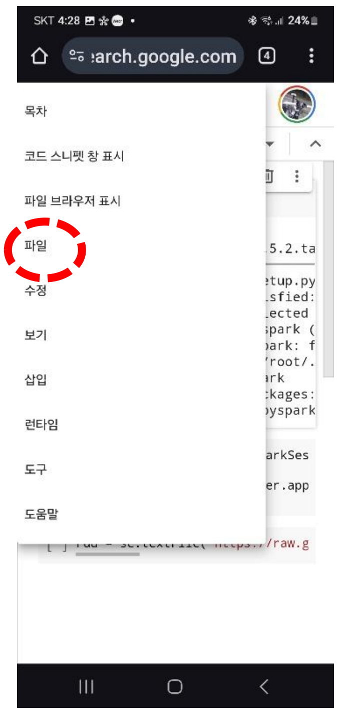
- Follow the qr link & open the file on you smart phone

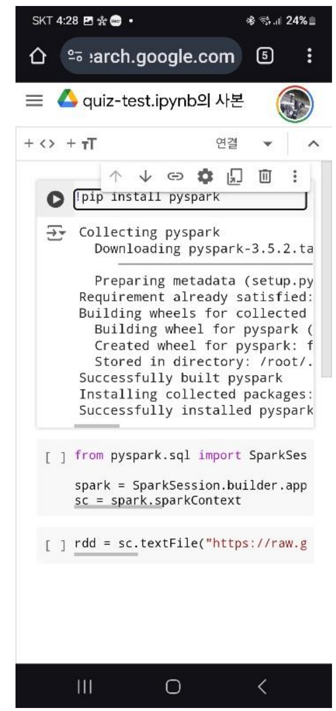


Copy Template File

- Make a copy and save on your drive on the directory “/Colab Notebook/”

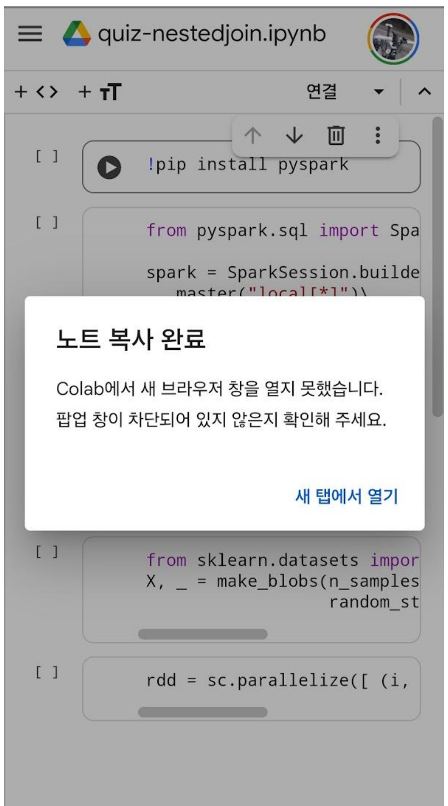


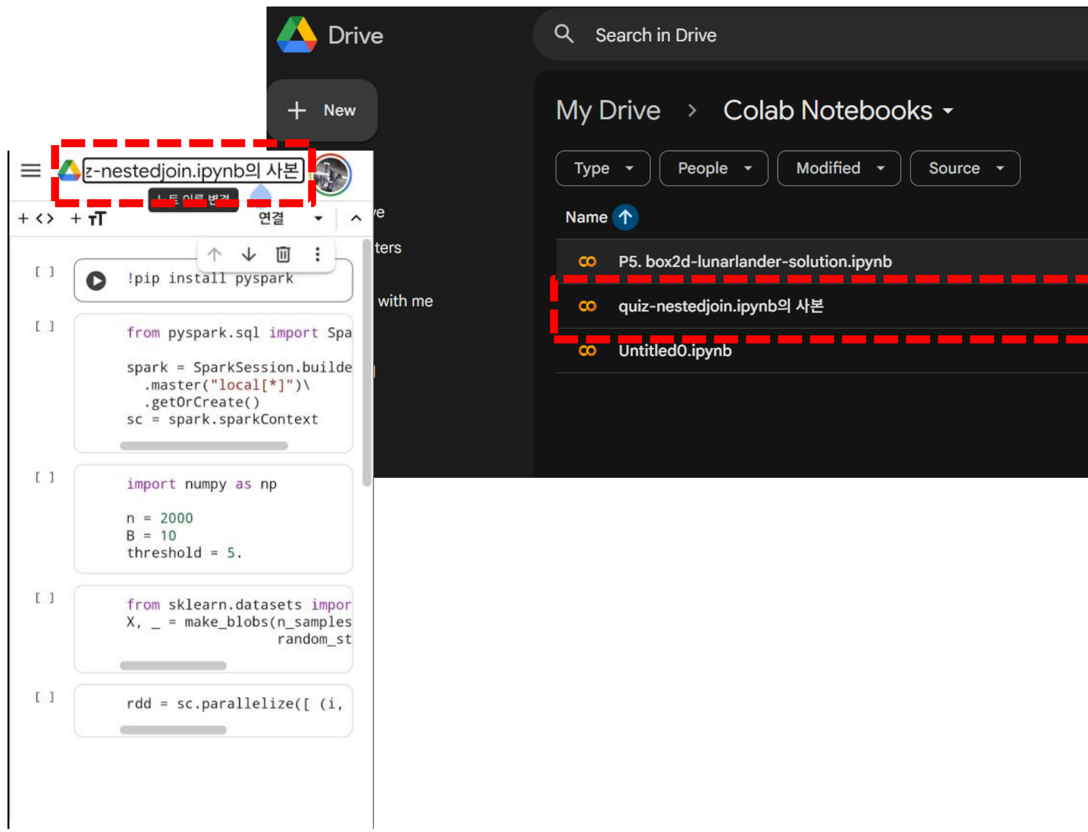




Complete Your Copy

- Complete the copied file on "/Colab Notebook"
- Make sure you do not edit the public template file





PySpark

- Complete the PySpark code to run nested loop join-like MapReduce process

Weak Points

Too large RDD: $O(n^2)$

With
points t_i,
< (1, i), t_i >

< (2, i), t_i >

…

< (i-1, i), t_i >

< (i, i+1), t_i >

…

< (i, n), t_i >

groupByKey

< (i, j), (t_i, t_j) >

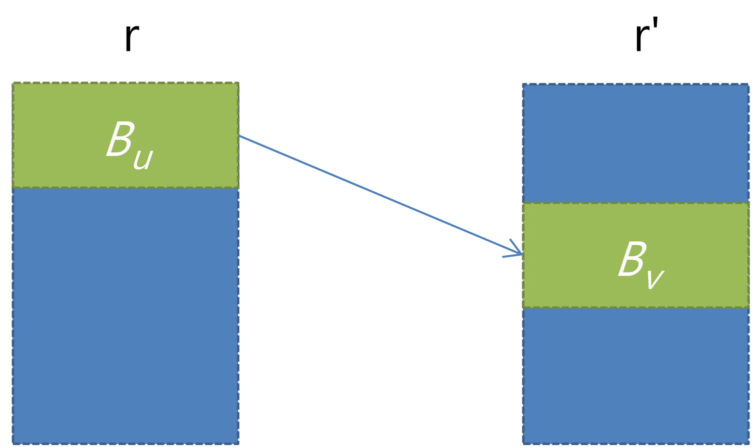
Too many function calls

filter

< (i, j), d(t_i, t_j) >

Block Nested Loop Self-Join

- **for each** block B_u **of** r **do begin**
  - **for each next** block B_v **of** r **do begin**
    - **for each** tuple t_i **in** B_u **do begin**
      - **for each** tuple t_j **in** B_v **do begin**
        - Check if (t_i, t_j) satisfy the join condition
        - if they do, add $t_i \bullet t_j$ to the result.
      - **end**
    - **end**
  - **end**
- **end**



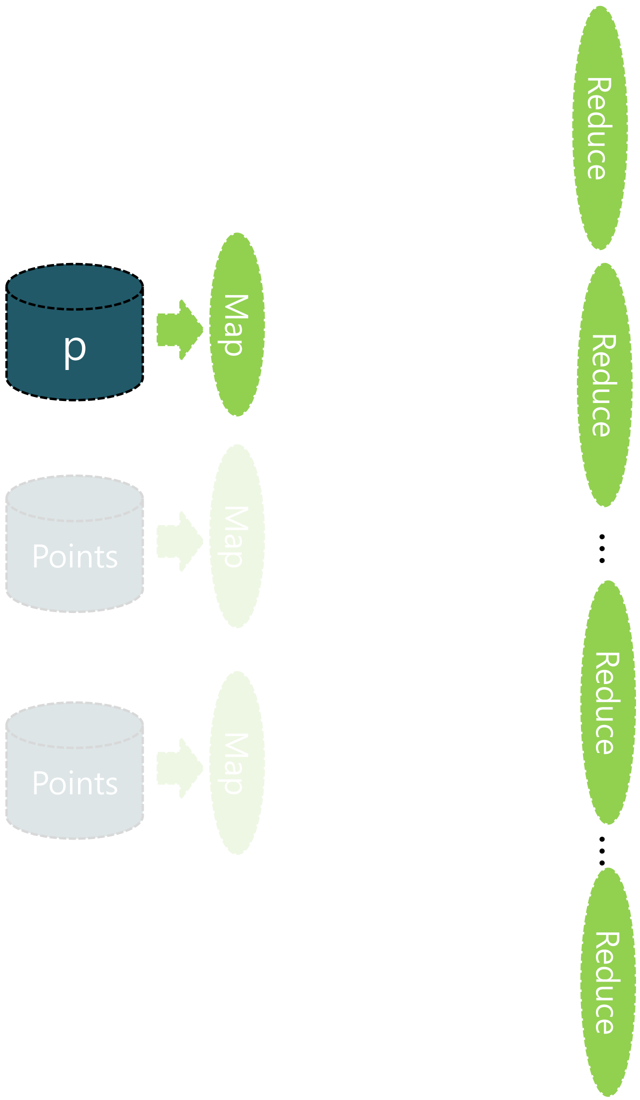
The reducer of key (Bu, Bv) computes the distance of all pairs from the blocks Bu and Bv

Thinking in MapReduce

1. Define mapper's input.
   - "ID, 1.2, 2.3"
2. Define reducer's key.
   - (u, v) where u and v are Block IDs, and u <= v
3. Define the key-value pairs in mapper.
   - Keys: (1, u), (2, u), …, (u, u), …, (u, m)
4. Define the output in reducer.
   - if u < v, compute join between B_u and B_v
     if u = v, compute self-join within B_u

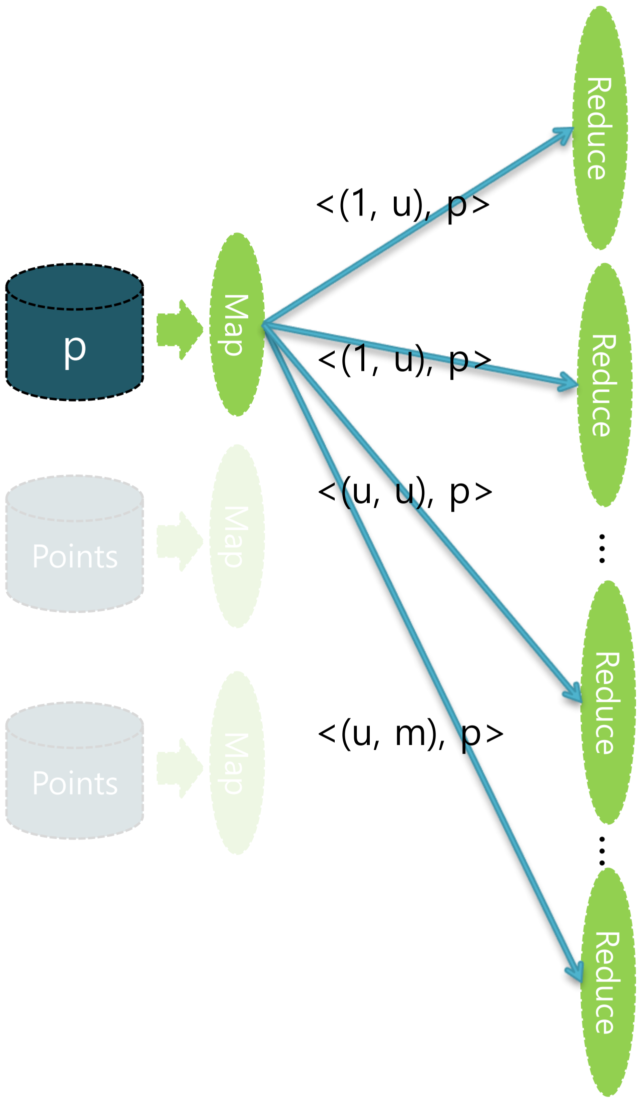
Parallel Block-nested Join

- Blocks
  - B_1, B_2, …, B_m : m distinct groups of records
- Map function's input
  - It takes a record p in the group u as input
- Reduce function's key
  - (u, v) : a partition pair to compute the similarities of all pairs of records from Bu and Bv (u ≤ v)

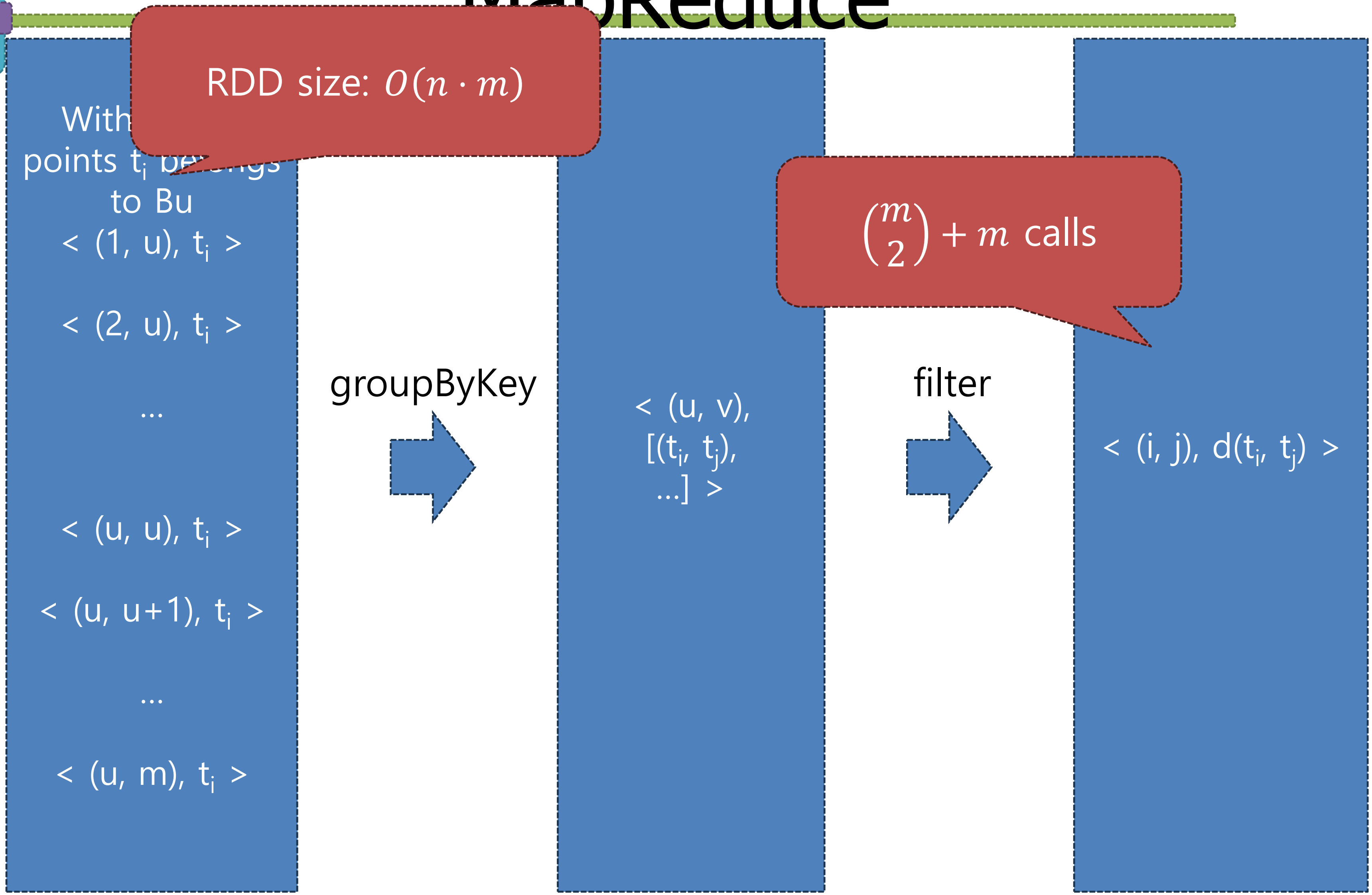


Parallel Block-nested Join

- Map function's output
  - For each record p in the group u, emit key-value pairs
    - <(1,u), p>,…,<(u, u), p>,…,<(u, m), p>
- Reduce function's output
  - (u, v) : a partition to compute the similarities of all pairs of records from Bu and Bv (u ≤ v)
    - u = v : self join in Bu
    - u < v : Cartesian join between Bu and Bv



Block Nested Loop Join-like MapReduce



Parallel Block-nested Self-Join

$$h(p_i) = \lceil i/2 \rceil$$

	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

Key	Value	Key	Value	Key	Value
(1,1)	p_1 = <0.78, 0.4, 0.01>	(1,2)	p_3 = <0.51, 0.11, 0.32>	(1,3)	p_5 = …
(1,2)	p_1 = <0.78, 0.4, 0.01>	(2,2)	p_3 = <0.51, 0.11, 0.32>	(2,3)	p_5 = …
(1,3)	p_1 = <0.78, 0.4, 0.01>	(2,3)	p_3 = <0.51, 0.11, 0.32>	(3,3)	p_5 = …
(1,1)	p_2 = <0.07, 0.21, 0.57>	(1,2)	p_4 = …	(1,3)	p_6 = …
(1,2)	p_2 = <0.07, 0.21, 0.57>	(2,2)	p_4 = …	(2,3)	p_6 = …
(1,3)	p_2 = <0.07, 0.21, 0.57>	(2,3)	p_4 = …	(3,3)	p_6 = …

(m=3) groups

Parallel Block-nested Self-Join

	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

(m=3) groups

Map

Key	Value list
(1,1)	p_1, p_2
(1,2)	p_1, p_2, p_3, p_4
(1,3)	p_1, p_2, p_5, p_6
(2,2)	p_3, p_4
(2,3)	p_3, p_4, p_5, p_6
(3,3)	p_5, p_6

Sort key-value pairs in the order of key & group by key

Reduce

Reduce

Threshold=0.6

Key	Value
(p_1, p_2)	0.92

Key	Value
(p_1, p_3)	0.50
(p_2, p_3)	0.52

Key	Value
(p_1, p_5)	0.02
(p_2, p_6)	0.04

Key	Value
(p_3, p_4)	0.92

Key	Value
(p_3, p_6)	0.49
(p_3, p_5)	0.50

Key	Value
(p_5, p_6)	0.05

Output pairs of data points with a distance closer than the threshold

Parallelization Efficiency

- 1) the total number of distance computation in all parallel reducers

  $= (n/m)\ (n/m-1)\ /\ 2 * m + (n/m) * (n/m) * m(m-1)/2$
  $=1 * 3\ \ + 4 * 3$
  $= n(n-1)\ /\ 2$
  $= 15$

- 2) with M workers: $O(n^2\ /\ M)$

Pseudocode of Map Function

- map_func (input)
  - x, vec = input
  - b = compute block ID of x
  - for i = 1 to n
    - if (i < b)
      - output <(i, b), (x, vec)>
    - else if (i >= x)
      - output <(b, i), (x, vec)>

Pseudocode of Reduce Function

- reduce (key, list)
  - x, y = key
  - if (x == y)
    - for each tuple (id_r, vec_r) in list
      - for each next tuple (id_s, vec_s) in list
      - check if (vex_r, vex_s) satisfy the join condition
        - if they do, output <(id_r, id_s), (vec_r, vec_s)>
  - else
    - list_r = vector list of block ID x in list
    - list_s = vector list of block ID y in list
    - for each tuple (id_r, vec_r) in list_r
      - for each tuple (id_s, vec_s) in list_s
        - check if (vex_r, vex_s) satisfy the join condition
        - if they do, output <(id_r, id_s), (vec_r, vec_s)>