Lightning Fast and Space Efficient Inequality Joins

Zuhair Khayyat, William Lucia, Megna Singh, Mourad Ouzzani, Paolo Papotti, Jorge Quiané, Nan Tang, Panos Kalnis

معهد قطر لبحوث الحوسبة
Qatar Computing Research Institute

Motivation

```
SELECT e1.id, e2.id
FROM Employees e1, e2
WHERE e1.salary > e2.salary
    AND e1.tax < e2.tax;
```

- **Important on many applications:** temporal and spatial databases, data cleaning, social networks, sensor networks,..
- **Inefficient solutions:** cross product + selection or expensive indexes

IEJoin

Algorithm 2: IESelfJoin

```
input : query Q with 2 join predicates t1.X op1 t2.X
        and t1.Y op2 t2.Y, table T of size n
output: a list of tuple pairs (ti, tj)
let L1 (resp. L2) be the array of column X (resp. Y)
if (op1 ∈ {>, ≥}) sort L1 in ascending order
else if (op1 ∈ {<, ≤}) sort L1 in descending order
if (op2 ∈ {>, ≥}) sort L2 in descending order
else if (op2 ∈ {<, ≤}) sort L2 in ascending order
compute the permutation array P of L2 w.r.t. L1
initialize bit-array B (|B| = n), and set all bits to 0
initialize join_result as an empty list for tuple pairs
if (op1 ∈ {≤, ≥} and op2 ∈ {≤, ≥}) eqOff = 0
else eqOff = 1
for (i ← 1 to n) do
    pos ← P[i]
    B[pos] ← 1
    for (j ← pos + eqOff to n) do
        if B[j] = 1 then
            add tuples w.r.t. (L1[j], L1[P[i]]) to
            join_result
return join_result
```

Employees

id	id	ssn	salary	tax
102	100	3456	100	10
104	101	9876	90	5
103	102	6790	160	14
100	103	4567	140	40
101	104	2095	150	15

Sort on Salary

id	salary	tax
102	160	14
104	150	15
103	140	40
100	100	10
101	90	5

Sort on Tax

id	salary	tax
103	140	40
104	150	15
102	160	14
100	100	10
101	90	5

PA

order_sal
2
1
0
3
4

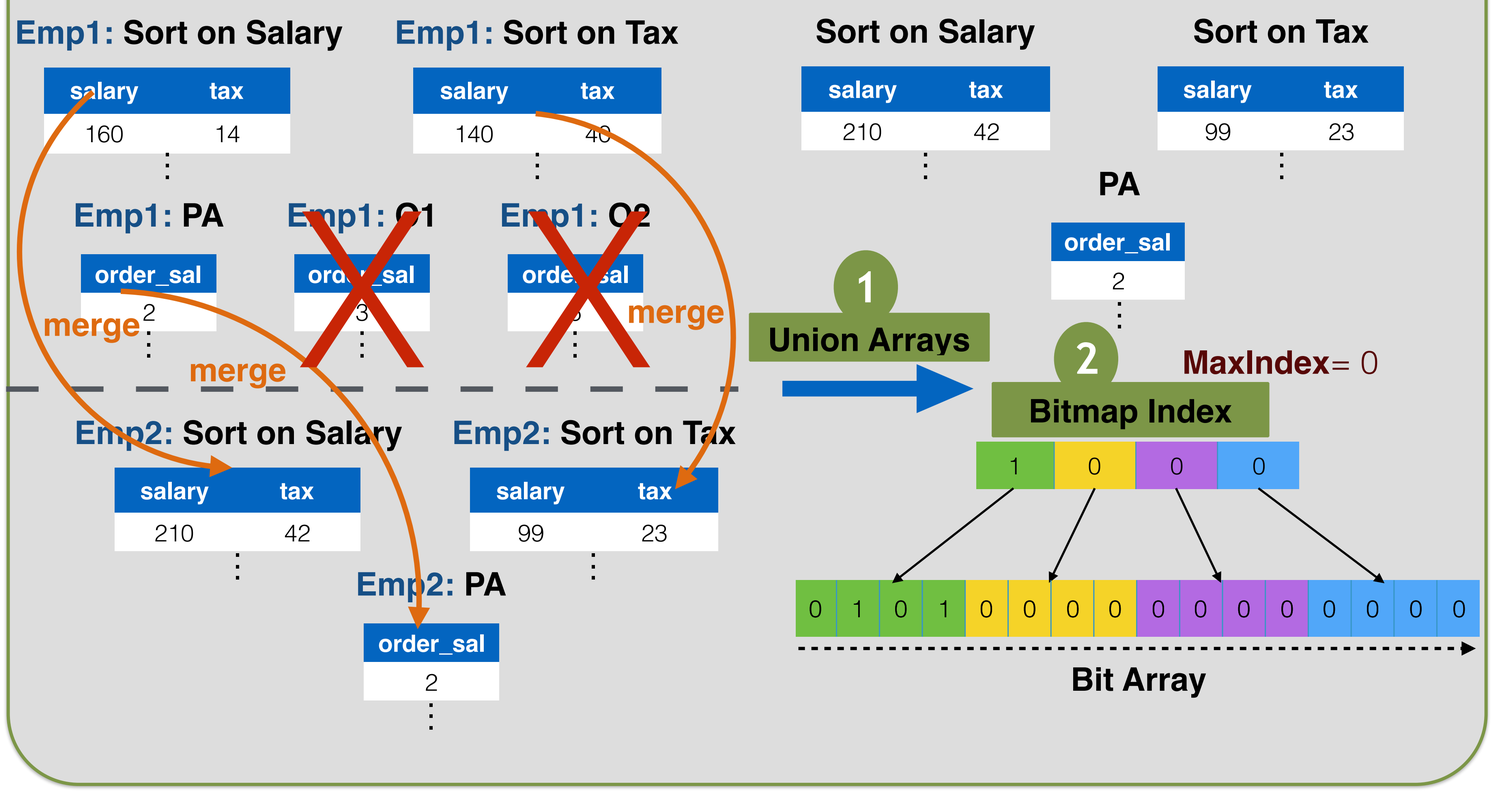
Results

e1.id	e2.id
104	103
102	104
102	103

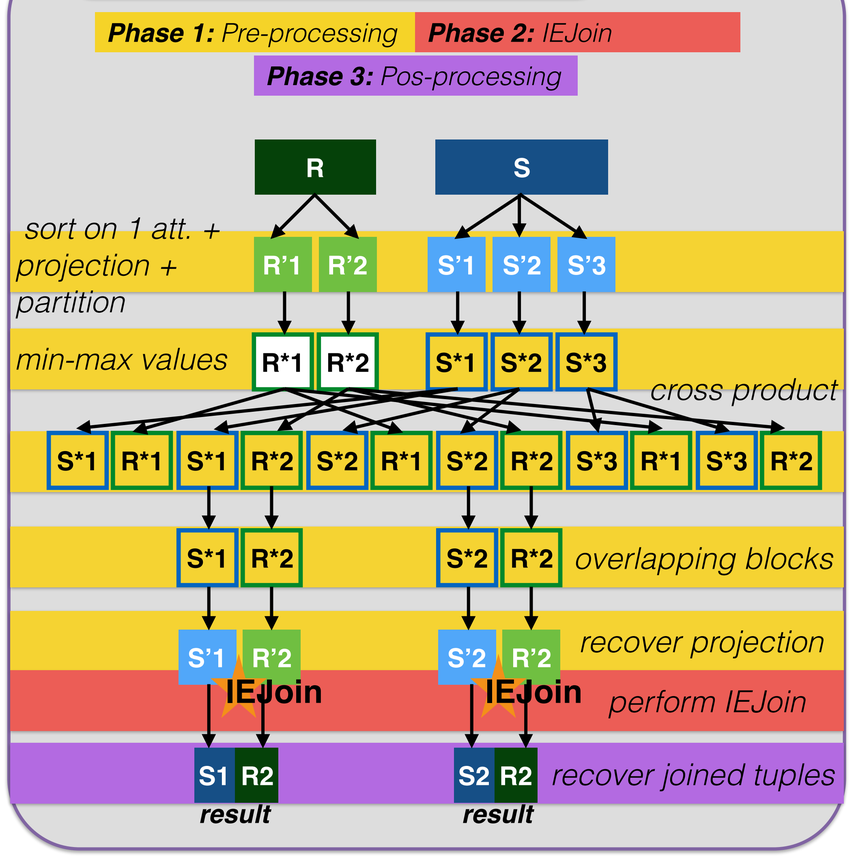
Bit Array

0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	1	1	0	0
1	1	1	1	0
1	1	1	1	1

Optimizations

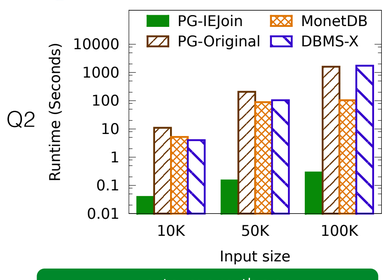


Scalable IEJoin



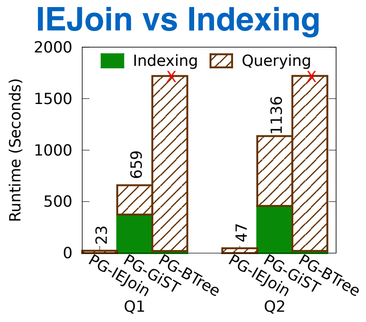
Results

Single-Node Performance



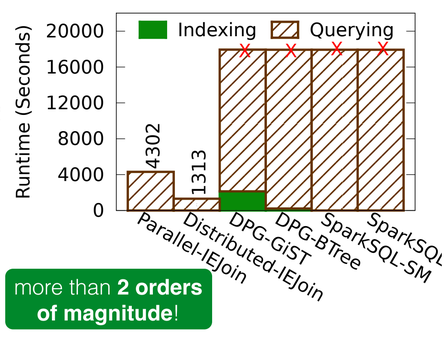
up to more than **3 orders of magnitude**!

IEJoin vs Indexing



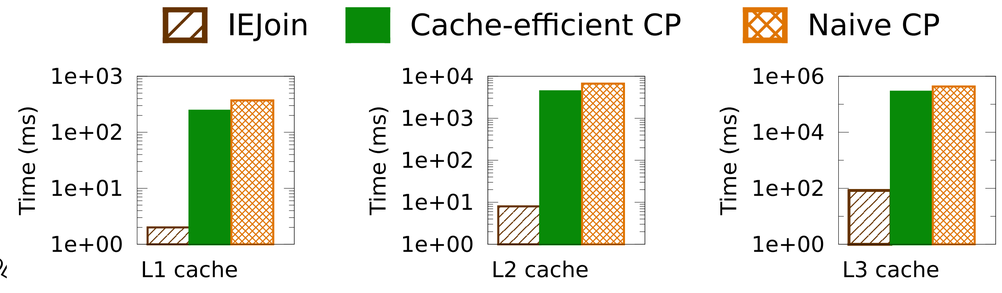
more than **1 order of magnitude**!

Multi-Node Performance



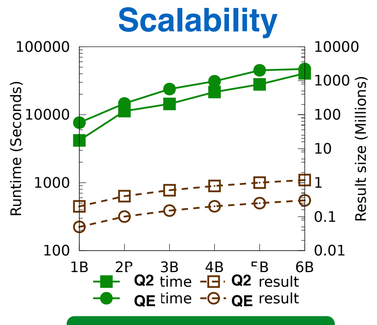
more than **2 orders of magnitude**!

Cache-Efficiency



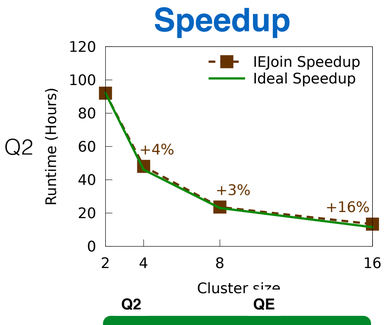
(a) L1 cache (b) L2 cache (c) L3 cache

Scalability



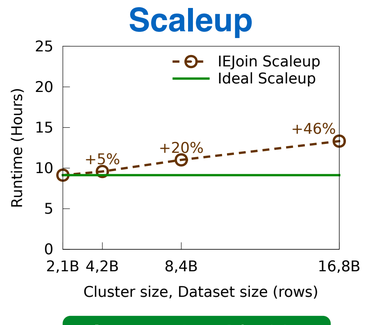
it scales **gracefully**

Speedup



close to **ideal** speedup

Scaleup



low overhead for big

Memory Consumption

Query	Input	Output	Time(secs)	Mem(GB)
Q_1	100K	9K	0.30	0.1
Q_1	**200K**	1.1K	0.5	**0.2**
Q_1	1M	29K	2.79	0.9
Q_1	10M	3M	27.64	8.8
Q_2	100K	0.2K	0.34	0.1
Q_2	**200K**	0.8K	0.65	**0.2**
Q_2	1M	20K	3.38	0.9
Q_2	10M	2M	59.6	9.7
Q_4	100K	6M	2.8	0.1
Q_4	200K	25M	10.6	0.2
Q_4	1M	0.4B	186	0.9
Q_4	**10M**	**50.5B**	**28,928**	**8.2**

low memory footprint

Highly **cache efficient**

Code available at:
https://github.com/daqcri/rheem

RHEEM