Leveraging Range Joins for the Computation of Overlap Joins

Anton Dignös[1], Michael H. Böhlen[2], Johann Gamper[1], Christian S. Jensen[3], and Peter Moser[4]

[1]Free University of Bozen-Bolzano, [2]University of Zurich, [3]Aalborg University, [4]NOI Techpark Südtirol/Alto Adige

Overlap Join

Given two relations containing periods, find pairs of tuples satisfying an **equality predicate** and **overlap on periods** [1].

```sql
SELECT *
FROM emp e JOIN dept d ON e.DNo = d.DNo AND e.P OVERLAPS d.P;
```

emp

		P	
EName	**DNo**	**[B**	**E)**
Sam	2	1	6
Ann	1	2	5
Joe	2	4	8
Sue	1	9	11

dept

		P	
DNo	**DName**	**[B**	**E)**
1	HR	1	11
2	Test	1	6
2	QA	6	10

Overlap join for equality on DNo and overlap on **P**

		P				P	
EName	**DNo**	**[B**	**E)**	**DNo**	**DName**	**[B**	**E)**
Sam	2	1	6	2	Test	1	6
Ann	1	2	5	1	HR	1	11
Joe	2	4	8	2	Test	1	6
Joe	2	4	8	2	QA	6	10
Sue	1	9	11	1	HR	2	11

Challenges

- Overlap predicate consist of inequalities on 4 attributes

  ⇒ Specialized algorithms/indices required

- Additional equality predicate needs to be supported

  ⇒ `WHERE _ = _ AND _ overlaps _`

- Different interval definitions should be supported

  ⇒ `[B,E), [B,E], (B,E), (B,E]`

Evaluation using Range Joins

1. Transformation of the overlap predicate

Equivalence:

$$r.P \text{ overlaps } s.P \equiv r.B < s.E \wedge s.B < r.E$$
$$\equiv r.B \leq s.B < r.E \vee s.B < r.B < s.E$$

Properties:

- Two disjunctive range conditions (≈ BETWEEN AND)
- Two conditions are disjoint (can be evaluated independently)

Rewrite:

```sql
SELECT *
FROM emp e JOIN dept d
     ON e.DNo = d.DNo AND e.B < d.E AND d.B < e.E;
```

As union (all) of two range joins:

```sql
SELECT *
FROM emp e JOIN dept d
     ON e.DNo = d.DNo AND e.B <= d.B AND d.B < e.E
UNION ALL
SELECT *
FROM emp e JOIN dept d
     ON d.DNo = e.DNo AND d.B <= e.B AND e.B < d.E
```

2a) Index-based evaluation

- Each range join can exploit and index

```sql
CREATE INDEX e_idx ON emp(dno, b);
CREATE INDEX d_idx ON dept(dno, b);
```

- Range join execution using an index-nested loop
- Append of the two range joins

```
                    QUERY PLAN
-----------------------------------------------------
Append

  -> Nested Loop
    -> Seq Scan on emp e
    -> Index Scan using d_idx on dept d
         Index Cond: ((dno = e.dno)
                      AND (e.b <= b) AND (b < e.e))

  -> Nested Loop
    -> Seq Scan on dept d_1
    -> Index Scan using e_idx on emp e_1
         Index Cond: ((dno = d_1.dno)
                      AND (d_1.b < b) AND (b < d_1.e))
```

- Works out of the box in DBMSs supporting B-trees

2b) Stand-alone range-join algorithm

- Sort-merge based algorithm for range joins

Algorithm 1: $\mathrm{RMJ}(\mathbf{r}, \mathbf{s}, \mathbf{C}, B, \prec^S, X, \prec^E, E, O)$

Input: Relation $\mathbf{r}$ sorted by $(\mathbf{C}, B)$, Relation $\mathbf{s}$ sorted by $(\mathbf{C}, X)$, equality attributes $\mathbf{C}$, start point B in $\mathbf{r}$, comparison operator $\prec^S \in \{<, \leq\}$ for B and X, attribute X in $\mathbf{s}$, comparison operator $\prec^E \in \{<, \leq\}$ for X and E, end point E in $\mathbf{r}$, output schema O

Output: Result of $\mathbf{r} \bowtie_{\mathbf{r}.\mathbf{C}=\mathbf{s}.\mathbf{C} \wedge \mathbf{r}.B \prec^S \mathbf{s}.X \prec^E \mathbf{r}.E} \mathbf{s}$.

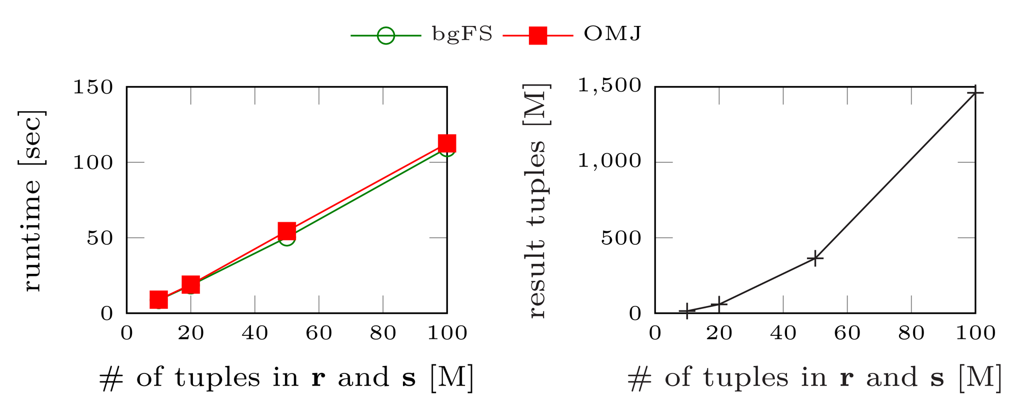
```
r ← first(r);
s ← first(s);
while r ≠ ω ∧ s ≠ ω do
    if r.C < s.C then
        r ← next(r);                          // skip outer
    else if r.C = s.C ∧ r.B ≺^S s.X then
        marked ← s;                           // mark
        while s ≠ ω ∧ r.C = s.C ∧ s.X ≺^E r.E do
            output r and s according to schema O;
            s ← next(s);
        r ← next(r);          // end of matches for outer
        s ← marked;                 // backtrack inner
    else
        s ← next(s);                          // skip inner
```

- PostgreSQL implementation available [2]

Experimental Evaluation

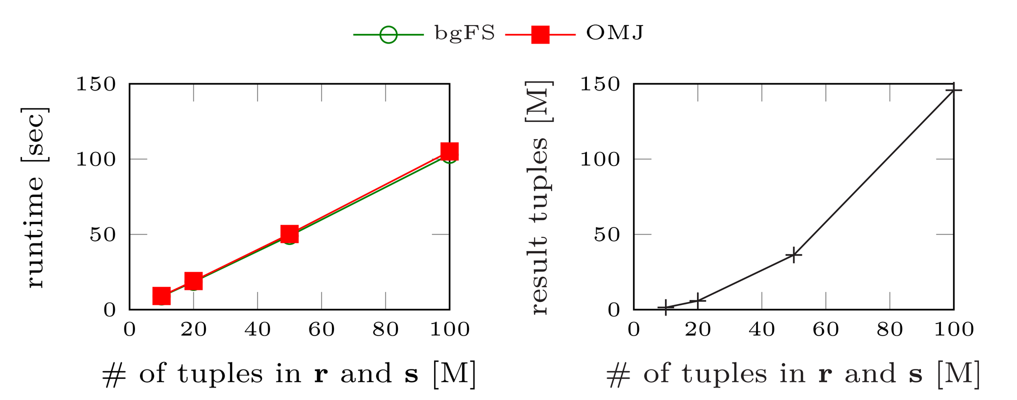
Stand-alone (main memory) algorithm

- Without equality condition



(a) Runtime (b) Output

- With equality condition

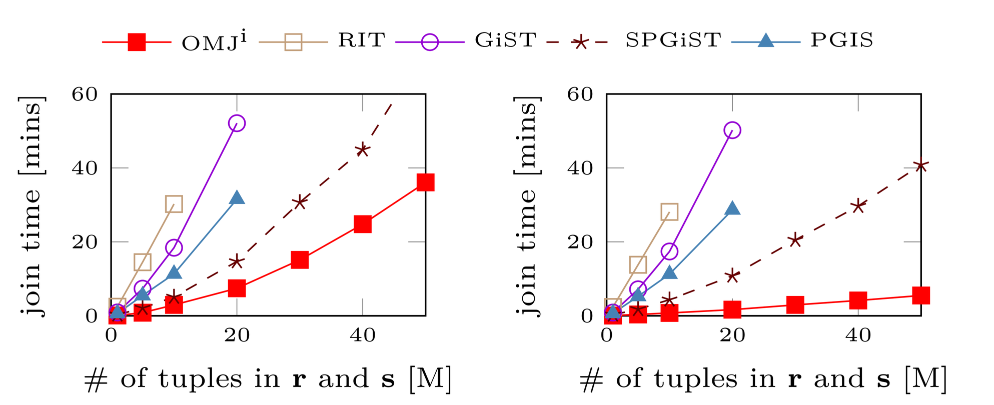


(a) Runtime (b) Output

- On par with the state-of-the art [3]
- More general algorithm

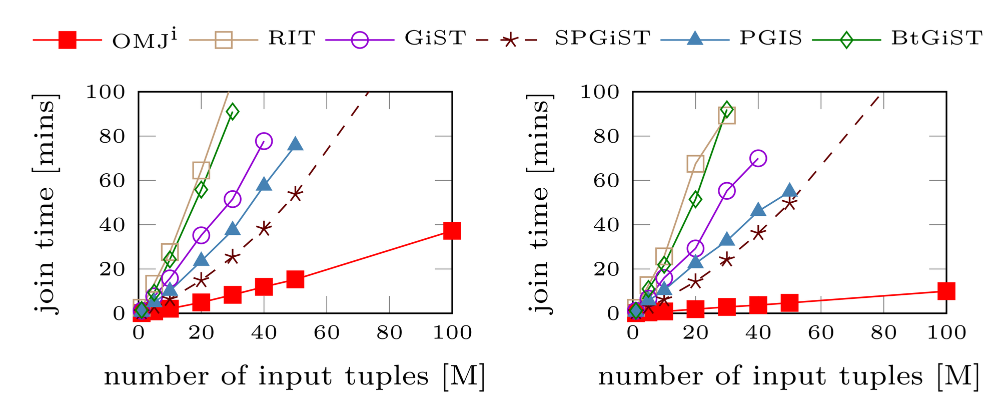
Index-based solution (in PostgreSQL)

- Without equality condition



(a) Nonclustered (b) Clustered

- With equality condition



(a) Nonclustered (b) Clustered

- Much faster compared to the state-of-the art (e.g., [4])
- Clustering is very effective

Summary of Contributions

- We provide a new and simple rewriting of the overlaps predicate that transforms an overlap join into the union of two independent range joins.
- Our solution supports the combination of the overlaps predicate with non-temporal equality constraints.
- We provide a strict total order for period boundaries over discrete and continuous domains and prove its correctness. This enables support for all common interval definitions for period timestamps as well as relations where tuples might have period timestamps with different interval definitions.
- We show how to evaluate overlap joins in DBMSs by taking advantage of B+-trees.
- We show how the rewriting can be used to devise an efficient yet simple main memory algorithm for overlap joins based on the sort-merge join paradigm.
- An extensive empirical evaluation shows that (a) our main memory algorithm performs on par with the state-of-the-art stand-alone competitors and that (b) the evaluation of the overlap join using B+-trees in an existing DBMS outperforms the state-of-the-art systems competitors.

References

[1] Kulkarni and Michels: Temporal features in SQL: 2011. SIGMOD Record 2012.

[2] `https://tpg.inf.unibz.it/project-rmj`

[3] Bouros and Mamoulis: A Forward Scan based Plane Sweep Algorithm for Parallel Interval Joins. PVLDB 2017.

[4] Enderle et al: Joining Interval Data in Relational Databases. SIGMOD 2004.

Funding

This work was funded by the Autonomous Province of Bozen-Bolzano Research "Südtirol/Alto Adige 2019" through the project Enabling Industrial-Strength, Open-Source Temporal Query Processing – ISTeP and by the Innovation Fund Denmark centre, DIREC.

Published in: *The VLDB Journal, Volume 31, Number 1, January 2022.*
Presented at: *48th International Conference on Very Large Data Bases*, Sydney, Australia (and hybrid), September 05–09, 2022.