Overlap Interval Partition Join

Anton Dignös¹  Michael H. Böhlen¹  Johann Gamper²

¹University of Zürich, Switzerland
²Free University of Bozen-Bolzano, Italy

SIGMOD 2014
June 22-27, 2014 - Snowbird, Utah, USA
Introduction

- **Temporal relations**: tuples have a time interval.
- **Overlap join**: join tuples with overlapping time intervals.

![Diagram showing temporal relations and overlap join]
Introduction

- **Temporal relations**: tuples have a time interval.
- **Overlap join**: join tuples with overlapping time intervals.

![Diagram showing temporal relations and overlap join]

- **Goal**: Efficient and robust overlap join
  - Alternative for query optimizer when other predicates are absent, have poor selectivity (long histories), or need to be evaluated after the join (on overlapping interval)
Outline

- \textit{OIP}: an efficient partitioning for interval data
- \textit{OIPJOIN}: a partition join based on \textit{OIP}
- Determine the optimal \textit{OIP} parameter $k$ for \textit{OIPJOIN}
- Empirical evaluation
Idea of Overlap Interval Partitioning \( OIP \)

- Given input data with intervals
Idea of Overlap Interval Partitioning (OIP)

- Given input data with intervals

- Partition intervals according to position and duration

- Constant clustering guarantee: Difference in duration of tuple and partition is upper-bounded by a constant.
Overlap Interval Partitioning (OIP)

- Divide time range into $k$ granules of equal duration
- Partitions are sequences of contiguous granules
- Partitions can overlap
Overlap Interval Partitioning (OIP)

- Divide time range into $k$ granules of equal duration
- Partitions are sequences of contiguous granules
- Partitions can overlap

$k = 3$: 

![Diagram showing partitioning of time range into three granules.](image)
Overlap Interval Partitioning (OIP)

- Divide time range into $k$ granules of equal duration
- Partitions are sequences of contiguous granules
- Partitions can overlap

$k = 3$: $Q$

$k = 4$: $Q$

Low $k$ $\Rightarrow$ fewer partition accesses (less overlapping boxes)

High $k$ $\Rightarrow$ more precise partitions (better fitting boxes)
Overlap Interval Partitioning (OIP)

- Divide time range into $k$ granules of equal duration
- Partitions are sequences of contiguous granules
- Partitions can overlap

$k = 3$:

Low $k \Rightarrow$ fewer partition accesses (less overlapping boxes)

$k = 4$:

High $k \Rightarrow$ more precise partitions (better fitting boxes)
The OIPJoin

1. Determine number of granules $k$
The OIPJoin

1. Determine number of granules $k$
2. Partition both input relations using $OIP$
The OIPJoin

1. Determine number of granules $k$
2. Partition both input relations using $OIP$
3. Join tuples within overlapping partitions
The OIPJoin

1. Determine number of granules $k$
2. Partition both input relations using $OIP$
3. Join tuples within overlapping partitions

Properties:
- Only 11 tuple comparisons
- 9 result tuples
- 2 false hits ($r_1 \circ s_6$ and $r_2 \circ s_5$)
- Only 5 inner partitions scanned (5 partition accesses)
Properties of OIP

- **Constant clustering guarantee**: The difference in duration between a tuple and its partition is less than two granules.
  - All tuples in a partition behave similarly
  - Very few false hits

- **Scans of partitions instead of random tuple access**:  
  - High cache locality
  - Much faster than index look-ups
How to Determine $k$?

Intuition: Find optimal $k$ s.t. the number of false hits of $OIP$ justifies the number of partition accesses and vice versa.
## Cost Dimensions

We consider CPU and IO costs

<table>
<thead>
<tr>
<th>Cost</th>
<th>CPU</th>
<th>IO</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Hits</td>
<td>Increase the number of CPU operations (identifying and discarding false hits).</td>
<td>Increase the number of block transfers (more data is fetched).</td>
</tr>
<tr>
<td>Partition Accesses</td>
<td>Increase the number of CPU operations (search in the access structure).</td>
<td>Increase the number of block transfers (more partially filled blocks)</td>
</tr>
</tbody>
</table>
Cost Dimensions

We consider CPU and IO costs

<table>
<thead>
<tr>
<th>Cost</th>
<th>CPU</th>
<th>IO</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Hits</td>
<td>Increase the number of CPU operations (identifying and discarding false hits).</td>
<td>Increase the number of block transfers (more data is fetched).</td>
</tr>
<tr>
<td>Partition Accesses</td>
<td>Increase the number of CPU operations (search in the access structure).</td>
<td>Increase the number of block transfers (more partially filled blocks)</td>
</tr>
</tbody>
</table>

What does that mean for $k$?

- **High** $k \Rightarrow$ **few** false hits, **many** partition accesses
- **Low** $k \Rightarrow$ **many** false hits, **few** partition accesses
Determining $k$ for the OIPJoin

1. Quantify false hits on average: $\text{AFR} \leq \frac{1}{k}$
   (Probability that a tuple is a false hit)

2. Quantify partition accesses on average: $\text{APA} = \frac{k^2 + k + 1}{3}$
   (Number of partitions accessed by a query interval)

3. Define the cost function for the overhead due to AFR and APA using CPU and IO cost

4. Minimize the cost function w.r.t. $k$
Overhead Cost for Partition Accesses

- For each of the $|p_r|$ outer partitions
  - APA inner partition accesses (scans)

  - partially filled blocks (1 trailing block per partition)
  - search in access structure (2 comparisons in access list)

  \[ |p_r| \cdot \text{APA} \cdot (c_{io} + 2 \cdot c_{cpu}) \]

- Average number of Partition Accesses APA = \( \frac{k^2 + k + 1}{3} \)
Overhead Cost for False Hits

- For each of the $|p_r|$ outer partitions
  - $\text{AFR} \cdot n_s$ false hits (inner) fetched
- Each outer tuple
  - Is compared with $\text{AFR} \cdot n_s$ false hits (inner)
  - Is $\text{AFR} \cdot n_s$ times a false hits

\[
|p_r| \cdot n_s \cdot \text{AFR} \cdot \frac{c_{io}}{b} + 2 \cdot n_s \cdot n_r \cdot \text{AFR} \cdot 2 \cdot c_{cpu})
\]

more data is fetched
(1 false hit within a block)

identifying and discarding
(2 comparisons per false hit)

- Average False hit Ratio $\text{AFR} \leq \frac{1}{k}$
The Overhead Cost Function

$$cost(k) = |p_r| \cdot APA \cdot (c_{io} + 2 \cdot c_{cpu}) +$$

- partially filled blocks (1 trailing block per partition)
- search in access structure (2 comparisons in access list)

- $|p_r| \cdot n_s \cdot AFR \cdot \left( \frac{c_{io}}{b} + 2 \cdot \frac{n_r}{|p_r|} \cdot 2 \cdot c_{cpu} \right)$
- more data is fetched (1 false hit within a block)
- identifying and discarding (2 comparisons per false hit)
- part. accesses false hits
Determining $k$ for the OIPJoin

- By minimizing $\text{cost}(k)$ we get:

$$k = f(n_r, n_s, c_{cpu}, c_{io}, b)$$

Example:

- $n_r = 10$M tuples
- $n_r = 100$M tuples
- $c_{cpu} = 0.5$
- $c_{io} = 10$
- $b = 15$ tuples on average in storage block

$$k = f(10\text{M}, 100\text{M}, 0.5, 10, 15) = 16,521$$
Related Work

- Overlap join based on **space partitioning** approaches, such as quadtree\(^1\) and loose quadtree\(^2\)
  - Divide time range recursively into two sub-ranges
  - Join cells of outer relation with all relevant of inner relation

- Properties
  - Long-lived tuples reside high up in hierarchy (many FH)
  - Cells grow with a factor of two (too much, many FH)
  - Parent cells are required for children (many possibly empty partitions)

- **OIPJOIN** does not deteriorate in performance with long-lived tuples, partitions grow by a constant factor.

---


Related Work /2

- Overlap join based on **indexing** approaches, such as interval tree, relational interval tree\(^3\), segment tree
  - Associate intervals with index node(s)
  - Join index nodes or tuples of outer relation with all relevant of inner

- Properties
  - Long-Lived tuples reside high up in hierarchy (\(~\) many partitions)
  - Requires many node joins (\(~\) many partitions)
  - No physical clustering possible (2 indices) (\(~\) FH in storage)

- **OIPJOIN** carefully balances the cost due to the access structure and groups tuple into partitions (cache locality)

---

Empirical Evaluation

1. Cost function compared with runtime

2. $k$ adapts to CPU and IO cost

3. Comparison with state-of-the-art approaches
   - Clustering guarantee is highly relevant for long-lived tuples
   - CPU cost is also relevant for disk resident data
Cost function Compared with Runtime

- **OIPJOIN** between 10M and 100M tuples
- Data in main memory

- Minimum of the cost function matches minimum of the runtime.
$k$ Adapts to CPU and IO Cost

Cost for access structure and false hits depends on CPU and IO cost.
Varying Duration of Tuples

- Outer and inner relation 10M tuples
- Data in main memory

- Clustering guarantee is important for long-lived tuples
- Partition scans more efficient than random memory access
Real World Datasets

- Personnel data
- File changes

Real world data contain a mix of short and long tuples
Varying Number of Tuples on Disk

- Outer relation 1% of inner relation
- Tuple durations up to 0.1%

Minimizing IOs is not enough

Also on disk the CPU cost of access structure and false hits is important.
Conclusion

Summary

- $OIP$ offers a constant clustering guarantee
- $OIPJoin$ is self-adjusting
- $OIPJoin$ outperforms state-of-the-art approaches

Future Work

- Advanced statistics to calculate the number of empty partitions for APA, e.g., using histograms.
- Study the maintenance of $OIP$.
- Refinement of cost function for different buffer replacement strategies.

SIGMOD 2014
Conclusion

Summary

- \( OIP \) offers a constant clustering guarantee
- \( OIP\text{JOIN} \) is self-adjusting
- \( OIP\text{JOIN} \) outperforms state-of-the-art approaches

Future Work

- Advanced statistics to calculate the number of empty partitions for APA, e.g., using histograms.
- Study the maintenance of \( OIP \).
- Refinement of cost function for different buffer replacement strategies.

Thank you for your attention!