Overlap Interval Partition Join

Anton Dignös[1] Michael H. Böhlen[1] Johann Gamper[2]

[1]University of Zürich, Switzerland

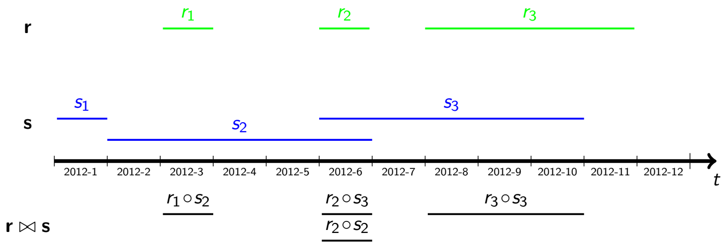
[2]Free University of Bozen-Bolzano, Italy

SIGMOD 2014

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

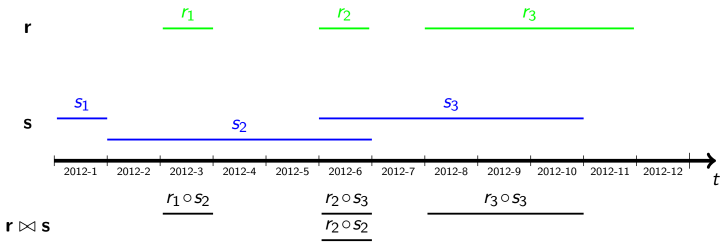
Introduction

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



Introduction

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



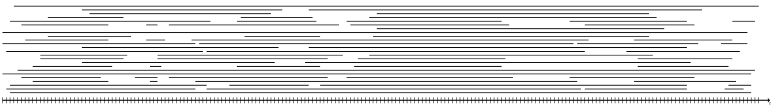
- **Goal**: Efficient and robust overlap join
  - Alternative for query optimizer when other predicates are absent, have poor selectivity (long histories), or need to be evaluated after the join (on overlapping interval)

Outline

- $\mathcal{OIP}$: an efficient partitioning for interval data
- OIPJoin: a partition join based on $\mathcal{OIP}$
- Determine the optimal $\mathcal{OIP}$ parameter k for OIPJoin
- Empirical evaluation

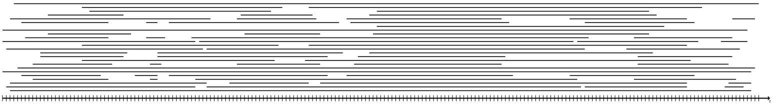
Idea of Overlap Interval Partitioning $\mathcal{OIP}$

- Given input data with intervals

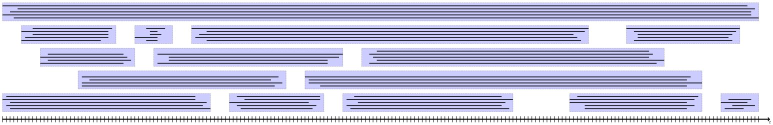


Idea of Overlap Interval Partitioning $\mathcal{OIP}$

- 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.

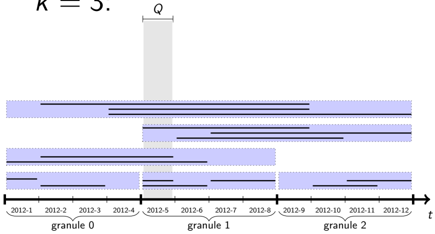
Overlap Interval Partitioning ($\mathcal{OIP}$)

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

Overlap Interval Partitioning ($\mathcal{OIP}$)

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

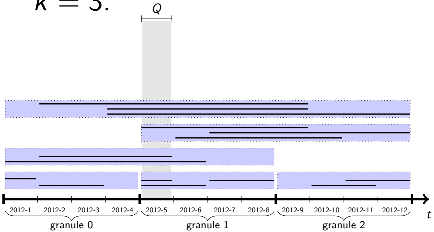
$k = 3$:



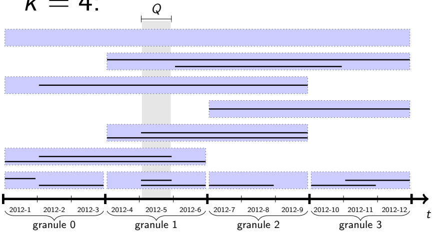
Overlap Interval Partitioning ($\mathcal{OIP}$)

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

$k = 3$:



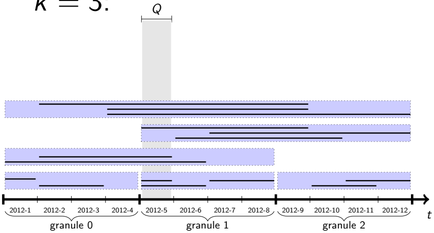
$k = 4$:



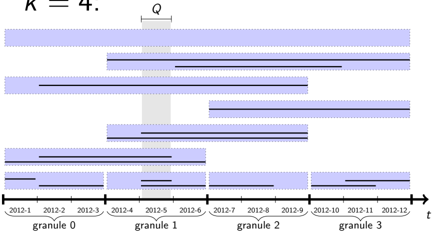
Overlap Interval Partitioning ($\mathcal{OIP}$)

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

$k = 3$:



$k = 4$:

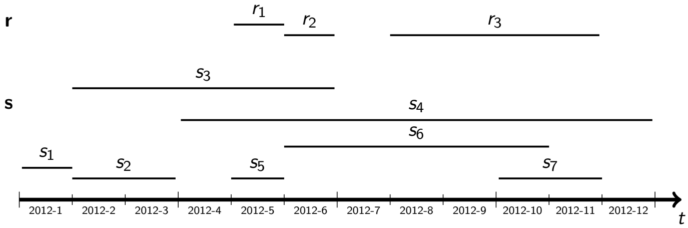


Low $k \Rightarrow$ fewer partition accesses (less overlapping boxes)

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

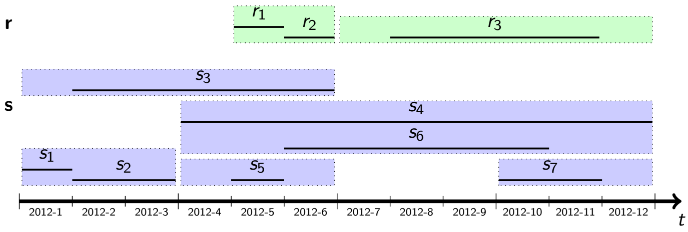
The OIPJoin

1. Determine number of granules k



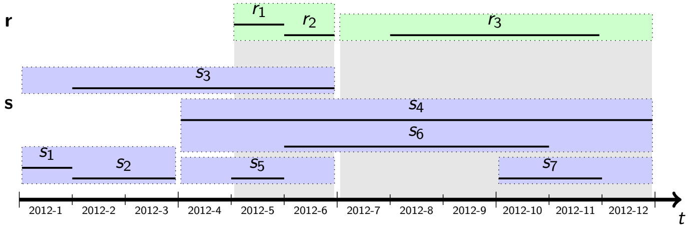
The OIPJoin

1. Determine number of granules k
2. Partition both input relations using $\mathcal{OIP}$



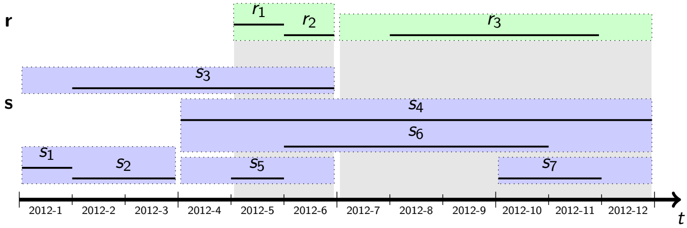
The OIPJoin

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



The OIPJoin

1. Determine number of granules k
2. Partition both input relations using $\mathcal{OIP}$
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 $\mathcal{OIP}$

- **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 $\mathcal{OIP}$ justifies the number of partition accesses and vice versa.

Cost Dimensions

We consider CPU and IO costs

Cost	CPU	IO
False Hits	Increase the number of CPU operations (identifying and discarding false hits).	Increase the number of block transfers (more data is fetched).
Partition Accesses	Increase the number of CPU operations (search in the access structure).	Increase the number of block transfers (more partially filled blocks)

Cost Dimensions

We consider CPU and IO costs

Cost	**CPU**	**IO**
False Hits	Increase the number of CPU operations (identifying and discarding false hits).	Increase the number of block transfers (more data is fetched).
Partition Accesses	Increase the number of CPU operations (search in the access structure).	Increase the number of block transfers (more partially filled blocks)

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: AFR $\leq \frac{1}{k}$
(Probability that a tuple is a false hit)

2. Quantify partition accesses on average: 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)

$$|p_r| \cdot \text{APA} \cdot (c\_io + 2 \cdot c\_cpu)$$ **part. accesses**

- 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

$$|p_r| \cdot n_s \cdot \boxed{\text{AFR}} \cdot \frac{c\_io}{b} + 2 \cdot n_s \cdot n_r \cdot \boxed{\text{AFR}} \cdot 2 \cdot c\_cpu) \qquad \textbf{false hits}$$

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

partially filled blocks
(1 trailing block per partition)

search in access structure
(2 comparisons in access list)

$$cost(k) = |p_r| \cdot \boxed{\text{APA}} \cdot (c\_io + 2 \cdot c\_cpu) + \quad \textbf{part. accesses}$$

$$|p_r| \cdot n_s \cdot \boxed{\text{AFR}} \cdot (\frac{c\_io}{b} + 2 \cdot \frac{n_r}{|p_r|} \cdot 2 \cdot c\_cpu) \quad \textbf{false hits}$$

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

identifying and discarding
(2 comparisons per false hit)

Determining k for the OIPJoin

- By minimizing $cost(k)$ we get:

$$k = f(n_r, n_s, c\_cpu, c\_io, b)$$

Example:

- $n_r = 10\text{M}$ tuples
- $n_r = 100\text{M}$ 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 /1

- 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.

[1] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. Acta Inf., 4:1-9, 1974.

[2] T. Ulrich. Loose octrees. In Game Programming Gems, pages 444-453. Charles River Media, 2000.

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 ($\sim$ many partitions)
  - Requires many node joins ($\sim$ many partitions)
  - No physical clustering possible (2 indices) ($\sim$ FH in storage)
- OIPJoin carefully balances the cost due to the access structure and groups tuple into partitions (cache locality)

---

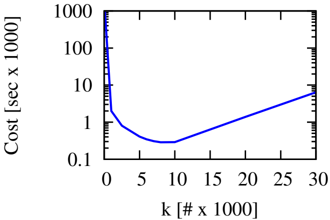
[3] H.-P. Kriegel, M. Ptke, and T. Seidl. Managing intervals efficiently in object-relational databases. In VLDB, pages 407418, 2000.
J. Enderle, M. Hampel, and T. Seidl. Joining interval data in relational databases. In SIGMOD, pages 683694, 2004.

Empirical Evaluation

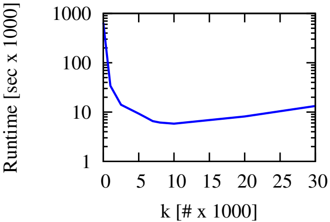
1. Cost function compared with runtime
2. *k* adapts to CPU and IO cost
3. Comparison with state-of-the-art approaches
   - Clustering guarantee is highly relevant for long-lived tuples
   - CPU cost is also relevant for disk resident data

Cost function Compared with Runtime

- OIPJoin between 10M and 100M tuples
- Data in main memory



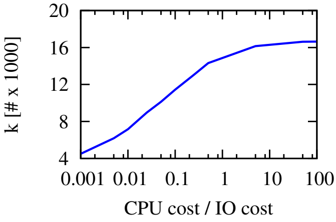
Cost Function

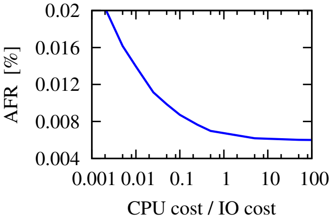


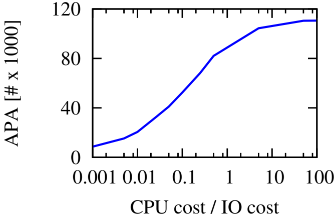
Runtime

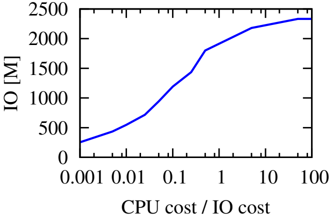
- Minimum of the cost function matches minimum of the runtime.

k Adapts to CPU and IO Cost





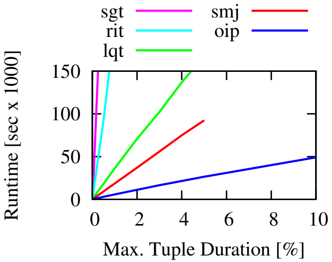


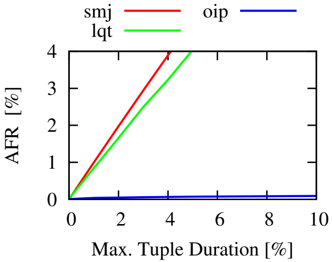


- Cost for access structure and false hits depends on CPU and IO cost.

Varying Duration of Tuples

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

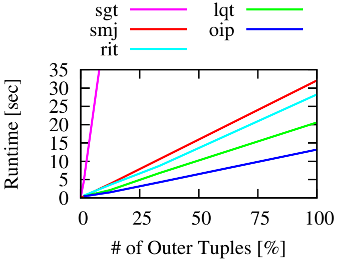




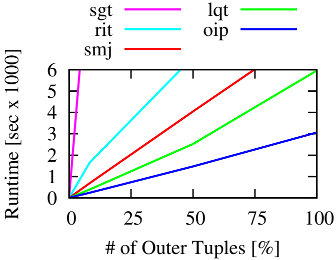
- Clustering guarantee is important for long-lived tuples
- Partition scans more efficient than random memory access

Real World Datasets

- Personnel data



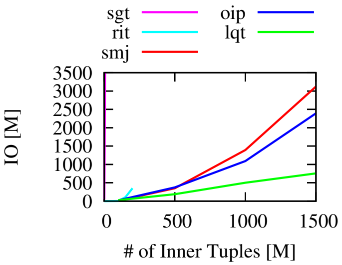
- File changes

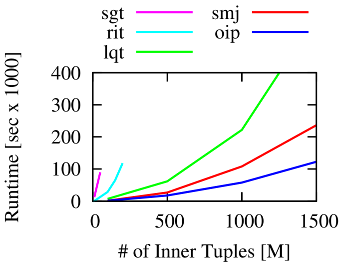


- Real world data contain a mix of short and long tuples

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

- $\mathcal{OIP}$ 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 $\mathcal{OIP}$.
- Refinement of cost function for different buffer replacement strategies.

Conclusion

Summary

- $\mathcal{OIP}$ 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 $\mathcal{OIP}$.
- Refinement of cost function for different buffer replacement strategies.

Thank you for your attention!