Ripple Joins for Online Aggregation

Parag Palekar,
Kashi Vishwanath

Motivation

- Generalize block nested-loops and hash-joins
- Update running estimates in a
  - smooth and
  - continuous fashion.

Motivation

- Previous best: Nested loops join [HHW97].
- \( R \times S, |R| < |S| \).
  - Pick a random \( s \) from \( S \)
  - Scan \( R \) to find all tuples <\( r, s \)>
  - Update running estimate and confidence interval.
- If \( R \) itself is of non-trivial size? Time between successive updates …
- Does length of successive intervals keep falling?

Ripple join: An overview

- A random tuple \( r \) from \( R \)
- A random tuple \( s \) from \( S \)
- Join \( r \) and \( s \) with previous tuples in \( R \times S \) and with each other.
  - Square version: equal rates for \( R \) and \( S \)
  - Rectangular version: unequal rates for each relation
- Tradeoff between sampling rate and confidence-length interval

A View of Ripple Join

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Square Ripple Join

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rectangular Ripple Join

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ripple Join contd…

- The general algorithm may contain \( K \) base relations, each retrieving \( \delta_k \) previously-unseen tuples from \( R_k \)
- The trade-off
  - If \( \delta_k \) values are higher
    - more number of I/Os are required…time between updates increases
    - length of the confidence interval is typically shorter
Ripple Join Algorithms

- Generalization of nested-loops join
- "inner" and "outer" loops keep continually interchanging.

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
</table>

Iterators

- Most DBMS's use an iterator model for relational operators.

<table>
<thead>
<tr>
<th>Types of iterator models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square Binary Ripple Join iterator</td>
</tr>
<tr>
<td>Enhanced Ripple Join iterator</td>
</tr>
</tbody>
</table>

An Enhanced Ripple Join Iterator

- Considers the non-unitary aspect ratios.
  - Retrieving tuples from two inputs of a ripple join at uneven rates.
  - An obvious solution is to maintain a local $Q$ value for each relation.

- Integration of a ripple join iterator into a query plan tree.
  - Input iterators must be restartable
  - Ripple joins can be combined with other join techniques

Ripple Join variants

- Block ripple join
  - Results in I/O savings factor proportional to the block size
- Index ripple join
  - Index on join attributes of $R$
  - No alternation between outer and inner loops
- Hash ripple join
  - Old tuples of $R$ and $S$ are maintained in memory and hashed on the join column

A view of Block ripple join

A block being considered for join instead of a tuple

| R | S |

Future Work

- Although ripple join is symmetric, it is unclear as to how
  - It should choose among ripple join variants
  - It should order a sequence of ripple joins
- Efficient processing of Self-joins
  - Avoiding two separate samples from the same input table
- Parallelization of ripple joins