Adaptive ordering of pipelined stream Filters
S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom
SIGMOD 2004

Presented by: Shimin Guo

Outline
- Introduction
- The Filter Ordering Problem
- The A-Greedy Algorithm
- The Sweep Algorithm
- The Independent Algorithm
- The LocalSwaps Algorithm
- Multiway Joins
- Experimental Evaluation
- Summary & Discussion

Introduction
- Streams processed by a set of commutative filters
- Overall processing costs depends on how the filters are ordered
- The best orderings are dependent on current stream and filter characteristics, which may change over time
- The selectivity of a filter depends on the filters before it
- Three-way tradeoff: convergence (C), run-time overhead (O), and speed of adaptivity (S)
The Filter Ordering Problem

- $n$ commutative filters: $F_1, F_2, \ldots, F_n$
- $f$: mapping from positions in the filters ordering to the indices of the filters at those positions
- $O$: an ordering $F_{f(1)}, F_{f(2)}, \ldots, F_{f(n)}$
- $d(i|j)$: conditional probability that $F_{f(j)}$ will drop a tuple, given that it was not dropped by any of $F_{f(1)}, F_{f(2)}, \ldots, F_{f(j-1)}$
- $t_i$: expected time for $F_i$ to process one tuple
- $D_i$: percentage of tuples that passed the first $i$ filters

The goal: minimize $\sum_{i=1}^{n} t_i D_i$

The A-Greedy Algorithm

- A greedy algorithm based on stable statistics:
  - Choose the filter $F_i$ with the highest $d(i|0)/t_i$ as the first filter.
  - Among the remaining filters, choose the filter $F_j$ with the highest $d(j|1)/t_j$ as the second filter.
  - And so on.
- Greedy Invariant (GI):
  \[
  \frac{d(i|j-1)}{t_{f(i)}} \leq \frac{d(j|j-1)}{t_{f(j)}} \quad 1 \leq i < j \leq n
  \]
- Goal of A-Greedy: maintain an ordering that satisfies the GI in an online manner

The A-Greedy Algorithm

- Two logical components of A-Greedy:
  - Profiler: continuously collects and maintains statistics about filter selectivities and processing costs
  - Reoptimizer: detects and corrects violations of the GI in the current filter ordering
- Challenge faced by the profiler: there are $n^{2^{n-1}}$ conditional selectivities for $n$ filters
- It’s impractical for the profiler to maintain online estimates of all these selectivities
- Solution: profile of recently dropped tuples
The A-Greedy Profiler

- Profile: a sliding window of profile tuples
- A profile tuple contains n boolean attributes \( b_1, \ldots, b_n \) corresponding to the n filters
- Dropped tuples are sampled with some probability, called drop-profiling probability
- If a tuple \( e \) is chosen for profiling, it will be tested by all remaining filters
- A new profile tuple inserted into the profile window, where \( b_i=1 \) if \( F_i \) drops \( e \) and \( b_i=0 \) otherwise

The A-Greedy Reoptimizer

- The reoptimizer maintains an ordering \( O \) such that \( O \) satisfies the GI for statistics estimated from the profile window
- How does the reoptimizer make use of the profile to derive estimates of conditional selectivities?
- It incrementally maintains a view over the profile window
The A-Greedy Reoptimizer

- View over the profile window: \( n \times n \) upper triangular matrix \( V \)
- \( V[i][j] \): number of tuples in the profile window that were dropped by \( F_{j(i)} \) but not dropped by \( F_{j(0)}, F_{j(2)}, \ldots, F_{j(i-1)} \)

\[
\begin{array}{c|cccc}
   & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 1 & 0 \\
4 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Profile window

\[
\begin{array}{c|cccc}
   & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Matrix view \( V \)

The A-Greedy Reoptimizer

- \( V[i][j] \) is proportional to \( d(j|i-1) \)
- Greedy Invariant:

\[
\frac{d(i|i-1)}{a(ji)} \leq \frac{d(j|i-1)}{a(ji)}, \quad \text{for } i, j \in \{1, \ldots, n\}
\]

\[
\frac{V[i][j]}{a(ji)} \leq \frac{V[j][i]}{a(ji)}, \quad \text{for } i, j \in \{1, \ldots, n\}
\]

(to avoid thrashing)

\[
\frac{V[i][j]}{a(ji)} \leq \alpha \frac{V[j][i]}{a(ji)}, \quad \text{for } i, j \in \{1, \ldots, n\}
\]
Convergence Properties

**Theorem 4.1.** When stream and filter characteristics are stable, the cost of a filter ordering satisfying the GI is at most four times the cost of the optimal filter ordering.

- Constant factor depends on number of filters, e.g., 2.35, 2.61, and 2.8 for 20, 100, and 200 filters, respectively
- Usually finds the optimal ordering in practice

Run-time Overhead

- Profile-tuple creation: needs additional $n-i$ evaluations for a tuple dropped by $F_{r0}$. Creation frequency determined by drop-profiling probability
- Profile-window maintenance: insertion and deletion of profile tuples. Also needs to maintain running averages of filter processing times
- Matrix-view update: every update would cause access to up to $n^2/4$ entries
- Violation detection: access to up to $n$ entries
- Violation correction: up to $n-i$ full scans of the profile window to correct a GI violation at position $i$

Speed of Adaptivity

- Any GI violation will be detected and corrected immediately
- Thus, A-Greedy is a very rapidly adapting algorithm
The A-Greedy Algorithm

- A-Greedy has good convergence properties and extremely fast adaptivity, but it imposes significant run-time overhead.
- Can we sacrifice some of A-Greedy’s convergence properties or adaptivity speed to reduce its run-time overhead?

The Sweep Algorithm

- Proceeds in stages.
- During one stage, only checks for GI violations involving the filter at one specific position $j$.
- Does not need to maintain the entire matrix view.
- Only $b_{f(j)}, \ldots, b_{f(j)}$ are required in the profile window.
- For each profiled tuple, needs to additionally evaluate $F_{f(j)}$ only.
- By rotating $j$ over $1, \ldots, n$, eventually detects and corrects all GI violations.

C, O, and S of Sweep

- Detects and corrects all GI violations $\rightarrow$ same convergence properties as A-Greedy.
- Reduced view and need for additional evaluations $\rightarrow$ less overhead.
- Only one filter is profiled in each stage $\rightarrow$ slower adaptivity.
The **Independent Algorithm**

- Assumes filters are independent
- Only needs to maintain estimates of unconditional selectivities

**C, O, and S of Independent**

- Convergence: dependent on whether assumption holds
  - if so, optimal
  - otherwise, can be $O(n)$ times worse than GI orderings
- Lower view maintenance overhead
- Fast adaptivity

The **LocalSwaps Algorithm**

- Monitors "local" violations only, i.e., violations involving adjacent filters
- Intuitively, LocalSwaps detects situations where a swap between adjacent filters in the current ordering would improve performance
- Only needs to maintain two diagonals of the view
- For each profiled tuple dropped by $F_{f_i}$, only needs to additionally evaluate $F_{f_{i+1}}$
C, O, and S of LocalSwaps

- Convergence: path-dependent
  - Best case: converges to GI orderings
  - Worse case: can be $O(n)$ times worse than GI orderings
  - May get stuck in local maxima
- Lower profiling and view-maintenance overhead
- Restricted to local moves → takes longer to converge

Comparison of the four algorithms

<table>
<thead>
<tr>
<th>A-Greedy</th>
<th>Sweep</th>
<th>Independent</th>
<th>LocalSwaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Good</td>
<td>Good</td>
<td>Path-dependent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Best case: same as A-Greedy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Worse case: $O(n)$ worse</td>
</tr>
<tr>
<td>O</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>S</td>
<td>Fast</td>
<td>Slow</td>
<td>Slow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiway Joins

- MJoins maintain an ordering of $\{S_0, S_1, \ldots, S_{n-1}\} \setminus \{S_i\}$ for each stream $S_i$
- New tuples arriving from $S_i$ is joined with other stream windows in that order
- Two-phase join algorithm
  - Drop-probing phase: the new tuple is used to probe all other windows in the specified order. If any window drops it, no further processing will be needed for it
  - Output-generation phase: if no window drops the tuple, proceeds as conventional MJoins
- Drop probing resembles pipelined filters
- A-Greedy and its variants can be used to determine the orderings
Multiway Joins

- Star Joins:
  - Tuple arrives from $S_0$: straightforward
  - Tuple arrives from $S_i$: join with $S_0$ first, then apply the two-phase join algorithm for each tuple in $S_i \times S_0$

- Acyclic Joins
  - Join graph defines a partial order
  - Join orderings constrained by the partial order

Experimental Evaluation

Convergence and overhead

![Graph showing convergence and overhead](image)

Experimental Evaluation

Speed of Adaptivity

- A-Greedy vs. Sweep
- A-Greedy vs. LocalSwaps
- A-Greedy vs. Independent
Experimental Evaluation
Varying the rate of change

Summary
- A-Greedy handles correlated filters
- A-Greedy has good convergence properties, fast adaptivity, but incurs significant run-time overhead
- Three variants of A-Greedy are proposed, each lying at a different point along the tradeoff spectrum among convergence, run-time overhead, and speed of adaptivity

Discussion
- Given that each of the algorithms has different utility in different settings, can we add another level of adaptivity that adaptively choose the algorithm that best fits the current setting?
- How in reality can correlation among filters affect query optimizers that assume independent filters?
- Is online reordering feasible for join operators that maintain internal states?
- How to choose the size for the profile window?