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

- 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: $F_1, F_2, \ldots, F_n$
- $f(\cdot)$: 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 $F_{f(i)}$ will drop a tuple e, given that e was not dropped by any of $F_{f(1)}, F_{f(2)}, \ldots, F_{f(j)}$
- t_i: expected time for F_i to process one tuple
- D_i : percentage of tuples that passed the first i-1 filters
- The goal: minimize $\sum_{i=1}^{n} t_{f(i)} D_i$

4

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):

$$\frac{d(i\,|\,i-1)}{t_{f(i)}} \geq \frac{d(j\,|\,i-1)}{t_{f(j)}},\ 1 \leq i \leq j \leq n$$

- Goal of A-Greedy: maintain an ordering that satisfies the GI in an online manner

5

The *A-Greedy* Algorithm

- Two logical components of A-Greedy:
  - profiler: continuouly 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 $n2^{n\text{-}1}$ 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

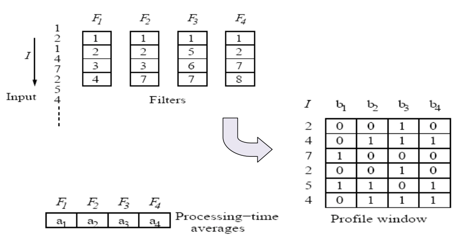
6

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

7

The *A-Greedy* Profiler



8

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

9

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 $F_{f(j)}$ but not dropped by $F_{f(1)}, F_{f(2)}, \ldots, F_{f(i\text{-}1)}$

I	b_1	b_2	b_3	b_4
2	0	0	1	0
4	0	1	1	1
7	1	0	0	0
2	0	0	1	0
5	1	1	0	1
4	0	1	1	1

Profile window

$\Rightarrow$

F_3	F_1	F_2	F_4
4	2	3	3
	2	1	1
		0	0
			0

Matrix view V

10

The *A-Greedy* Reoptimizer

- $V[i, j]$ is proportional to $d\,(j|i\text{-}1)$
- Greedy Invariant:

$$\frac{d(i\,|\,i-1)}{t_{f(i)}} \geq \frac{d(j\,|\,i-1)}{t_{f(j)}},\ 1 \leq i \leq j \leq n$$

$$\Downarrow$$

$$\frac{V[i,i]}{a_{f(i)}} \geq \frac{V[i,j]}{a_{f(j)}},\ 1 \leq i \leq j \leq n$$

$\Downarrow$ (to avoid thrashing)

$$\frac{V[i,i]}{a_{f(i)}} \geq \alpha \frac{V[i,j]}{a_{f(j)}},\ 1 \leq i \leq j \leq n$$

11

The *A-Greedy* Reoptimizer

```
/** Input: O violates the Greedy Invariant at position i */
maxr = i;
for (r = i; r ≤ maxr; r = r + 1) {
    /** Greedy Invariant holds at positions 1, ..., r − 1 and at
     *  maxr + 1, ..., n. Compute V entries for row r */
    for (c = r; c ≤ n; c = c + 1)
        V[r, c] = 0;
    for each profi le tuple ⟨b1, b2, ..., bn⟩ in the profi le window {
        /** Ignore tuples dropped by F_f(1), F_f(2), ..., F_f(r−1) */
        if (b_f(1) == 0 and b_f(2) == 0 and ··· and b_f(r−1) == 0)
            for (c = r; c ≤ n; c = c + 1)
                if(b_f(c) == 1) V[r, c] = V[r, c] + 1;
    }
    /** Find the column maxc with maximum V[r,maxc] / t_f(maxc) */
    maxc = r;
    for (c = r + 1; c ≤ n; c = c + 1)
        if (V[r, c]/a_f(c) > V[r, maxc]/a_f(maxc)) {
            maxc = c;
            if (maxc > maxr)   maxr = maxc;
        }
    if (r ≠ maxc) {
        /** Current fi lter F_f(maxc) becomes the new F_f(r).
         *  We swap the fi lters at positions maxc and r */
        for (k = 0; k ≤ r; k = k + 1)
            Swap V[k, r] and V[k, maxc]; }
```

Figure 5: Correcting a violation of the Greedy Invariant

rebuilding matrix view

swapping filters

12

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 $n\text{-}i$ evaluations for a tuple dropped by $F_{f(i)}$. 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 $n\text{-}i$ 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

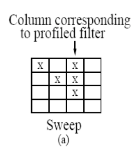
The *A-Greedy* Algorithm

- 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?

16

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 $b_{f(1)},\dots,b_{f(j)}$ are required in the profile window
- For each profiled tuple, needs to additionally evaluate $F_{f(j)}$ only
- By rotating j over $2,\dots,n$, eventually detects and corrects all GI violations



17

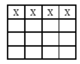
C, O, and S of *Sweep*

- Detects and corrects all GI violations
  → same convergence properties as A-Greedy
- Reduced view and need for additional evaluations
  → less overhead
- Only one filter is profiled in each stage
  → slower adaptivity

18

The *Independent* Algorithm

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



Independent
(b)

19

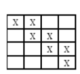
C, O, and S of *Independent*

- 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

20

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 $F_{f(i+1)}$



LocalSwaps
(c)

21

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

22

Comparison of the four algorithms

	A-Greedy	Sweep	Independent	LocalSwaps
C	Good	Good	Optimal if independence assumption holds, $O(n)$ worse in general	Path-dependent Best case: same as A-Greedy Worse case: $O(n)$ worse
O	High	Low	Low	Low
S	Fast	Slow	Fast	Slow

23

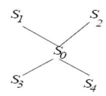
Multiway Joins

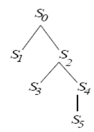
- MJoins maintain an ordering of $\{S_0, S_1, \dots, S_{n-1}\} - \{S_i\}$ for each stream S_i
- New tuples arriving from S_i is joined with other stream windows in that order
- Two-phase join algorithm
  - Drop-probing phase: the new tuple is used to probe all other windows in the specified order. If 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

24

Multiway Joins

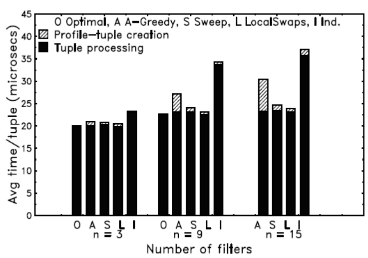
- Star Joins:
  - Tuple arrives from S_0: straightforward
  - Tuple arrives from S_i : join with S_0 first, then apply the two-phase join algorithm for each tuple in $s_i \bowtie S_0$
- Acyclic Joins
  - Join graph defines a partial order
  - Join orderings constrained by the partial order





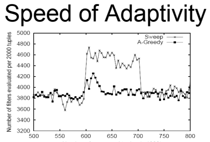
Experimental Evaluation

Convergence and overhead

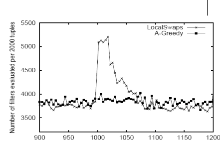


Experimental Evaluation

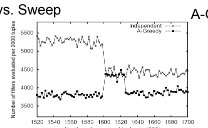
Speed of Adaptivity



A-Greedy vs. Sweep



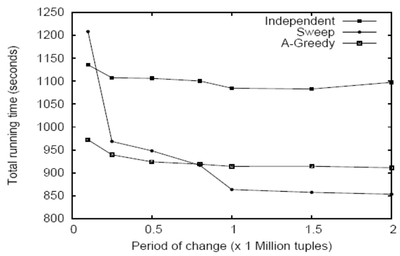
A-Greedy vs. LocalSwaps



A-Greedy vs. Independent

Experimental Evaluation

Varying the rate of change



28

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 diffenrent points along the tradeoff spectrum among convergence, run-time overhead, and speed of adaptivity

29

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?

30