# Scalable Approximate Query Processing With The DBO Engine

C. Jermaine et al.
Presented by: Yi Zhang
Jan 29, 2008

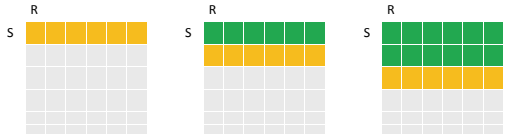
1

## Problem

- Need for online estimation of query answer
  - Updates as query processing progresses
  - Statistically meaningful bounds
  - Accurate final answer of course
  - Application:
    - Interactive data exploration over large datasets
- Query style
  - `SELECT FROM WHERE GROUP BY`

2

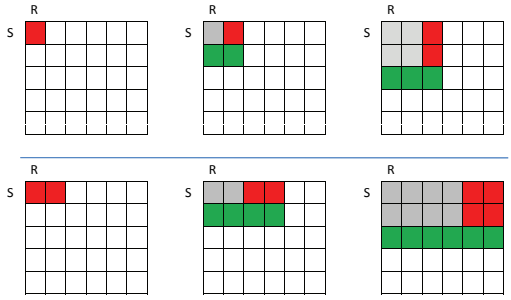
## Nested Loop Join




- Long delay between successive updates
- Limited contribution to shortening confidence interval

3

## Ripple Join




4

## Estimation & Statistics

- An example
  - `SELECT SUM(expression) FROM R,S WHERE p`
  - Estimated by $\frac{|R|\cdot|S|}{|R_o|\cdot|S_o|}\sum_{(r,s)\in R_o\times S_o}\text{expression}_p(r,s)$
- Estimator unbiased, consistent
- Capable of giving tight confidence intervals
- Variance can also be calculated
- But…
  - When samples go out of memory…
  - Significant disk I/Os


5

## SMS Join

- Generalization of ripple join
  - Identical to ripple join when all fits in memory
- Continues to provide confidence bounds when operating on disk
- Phases
  - Sort: read input relations in parallel, hash ripple join, and sort on join attributes
    - Interleave reads and writes across runs
  - Merge: similar to SMJ, parallel with Shrink
  - Shrink: update estimator after removing merged data space

6

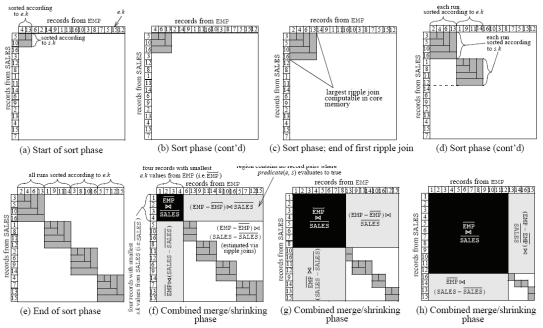



**Figure 4: Outline of an SMS join, assuming enough main memory to buffer four records from each input relation.**

7

# Scalability!

- Two phase model of sort-merge does not scale well

```
SELECT SUM(T.C)
FROM R,S,T
WHERE R.A=S.A AND S.B=T.B
```

- Fixes
  - Two separate joins: Can we do better?
  - Pipelining
    - Bad randomness
    - Hard to estimate e.g. intermediate relation size

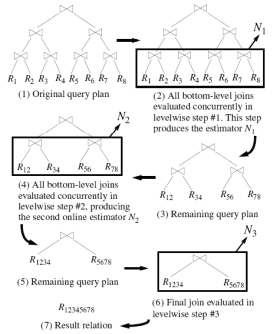
8

# How DBO Does It

- We need both scalability and progressive accurate estimation
- Design differences from traditional databases
  - Information sharing across relational operations
  - Processing abstracted into levelwise steps
  - Each levelwise step maintains and updates an online estimator
  - Results used as input for next levelwise step
  - Levelwise estimators combined with all lower level estimators to produce a single estimator for final answer

9

# A Quick Overview




10

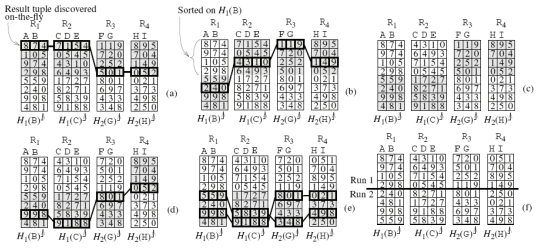
# Levelwise Step

- Scan & Merge
- Scan phase
  - Concurrent for input relations in a levelwise step
  - Records of a relation divided into *runs*
  - In-memory join for unbiased guess
  - Randomized sort order
    - Sort on H(id, R.key)
  - Round-robin processing of runs

11

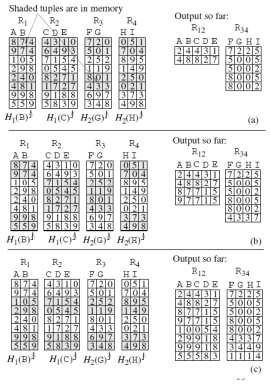
# Example

- …WHERE R1.B=R2.C AND R2.E=R3.F AND R3.G=R4.H




12

## Merge Phase




## Estimation

- First levelwise step
  - Sum of aggregate of tuples discovered so far * size ratio

$$\alpha = \sum_{a=1}^{r}\left[\sum_{t_a \in T(a,1,n)} f(t)\right] + \sum_{r=2}^{p}\sum_{a=1}^{r-1}\left[\sum_{t_1 \in T(a,1,b)}\sum_{t_2 \in T(a-1,b+1,n)} f(t_1 \bullet t_2)\right]$$

$$\beta = \frac{|R_1 \times R_2 \times \ldots \times R_n|}{\sum_{a=1}^{p}|T(a,1,n)| + \sum_{a=2}^{p}\sum_{b=1}^{a-1}|T(a,1,b)||T(a-1,b+1,n)|}$$

- What about subsequent levels?
  - Intermediate results grouped by join key – not completely random
  - Cardinality of intermediate results unknown due to pipelining

14

## Solution to Semi-Randomness

- "Clumps"
  - View group of tuples w/ same join key from a merge phase as a single, indivisible output tuple
  - Estimate using clumps

$$f(t_1 \bullet t_2 \bullet \ldots \bullet t_n) = \sum_{t_1' \in t_1}\sum_{t_2' \in t_2}\cdots\sum_{t_n' \in t_n} f(t_1' \bullet t_2' \bullet \ldots \bullet t_n')$$

15

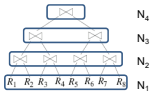
## Solution to Unknown Relation Size

- We don't need to estimate the unknown size!
- Partition H's output space into p equi-sized ranges
  - Size ratio $\beta = \frac{p^n}{1+(r-1)n}$
  - Round-robin scan searches 1+(r-1)n combinations of runs


16

## Putting All Estimators Together

- d+1 unbiased estimators for d leve
- Take weighted average




$$N = \sum_{i=1}^{d+1} w_i N_i$$

$$\sigma^2(N) = \sum_{i=1}^{d+1} w_i^2 \sigma^2(N_i)$$

Different hash seeds for different levels, so estimators independent

- To choose weights:
  - Lagrangian multiplier: $f(w_1, \cdots, w_n, \lambda) = \sigma^2(N) + \lambda(\sum_{i=1}^{n} w_i - 1)$
  - Variance minimized by

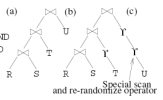
$$w_i = 1 \Big/ \left\{\sigma^2(N_i)\sum_{j=1}^{d+1}\frac{1}{\sigma^2(N_j)}\right\}$$

17

## Other Cases

- Other aggregates
  - GROUP BY: separate estimators for each group
  - So far only SUM aggregate considered
    - COUNT: also a simple version of SUM
    - AVG: function of SUM and COUNT
    - STD_DEV: function of AVG and COUNT
- Non-bushy query plan trees

```
SELECT SUM (R.A)
FROM R, S, T, U
WHERE R.A = S.A AND
      R.B = T.B AND
      R.C = U.C
```



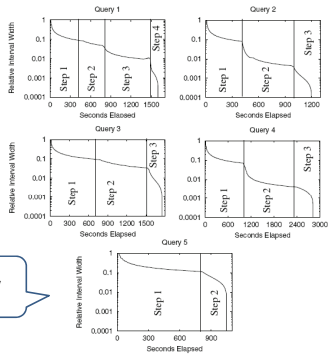

18

# Experiments

- Five queries over TPC-H schema
- 10GB generated data



Decrease of confidence interval width




y axis: width of 95% confidence interval / current estimate

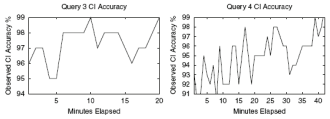


# Running Time and Accuracy

Query Execution Time

| Query | DBO | Postgres |
|---|---|---|
| $Q_1$ | 26m42s | 43m47s |
| $Q_2$ | 20m08s | 34m27s |
| $Q_3$ | 29m12s | 37m40s |
| $Q_4$ | 47m05s | 88m28s |
| $Q_5$ | 17m28s | 46m31s |

Fraction of confidence intervals that do contain actual answer






# Problems and Discussion

- Be organized or random?
  - Randomness provides unbiased online estimation
  - Indexing provides fast access
  - Can we make use of index? Can we access random data fast?
- Current system limits itself in many aspects
  - Sort-merge style join
  - Equality predicates
  - Are we losing the opportunity of optimization by this specific parallel way of query evaluation?
  - …