15-721 DB Sys. Design & Impl.

R* Optimizer

Christos Faloutsos
www.cs.cmu.edu/~christos

Roadmap

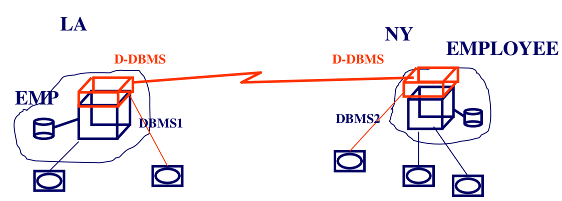
1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
3) Transactions: locking, recovery
4) Distributed DBMSs
   intro
   R* architecture
   ➡ R* optimizer
5) Parallel DBMSs: Gamma, Alphasort
6) OO/OR DBMS
7) Data Analysis - data mining

Citation

Mackert & Lohman, "*R* Optimizer Validation and Performance Evaluation for Distributed Queries*" VLDB 1986.

Problem – definition

ideally: **connect to distr-LA; exec sql select * from EMPL;**



Overview

- Distr. computation and Optimization
- Instrumentation
- Distr. join results
- Alternative join strategies
- Conclusions

Introduction - targets of study

- Validation of q-optimizer
  - how often is the chosen plan sub-optimal?
  - which are the most influential parms?
  - sensitivity analysis
- Other questions
  - other improvements for distr. joins?
  - other promising techniques?

Distr. comp. & opt.

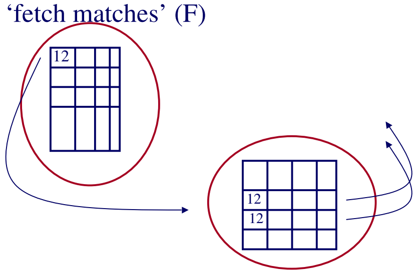
- Definitions / assumptions
  - each table: stored at one site (no partitioning)
  - dfn: query site; master site; apprentice sites

Distr. comp. & opt.

- in centralized q-opt:
  - join order; method; access path
- now: ++ join site
- how to send inner rel. to join site?
  - "ship whole" (W)
  - 'fetch matches' (F) (~ semijoin)

Distr. comp. & opt.

- 'fetch matches' (F)



Overview

- Distr. computation and Optimization
- ➡ Instrumentation
- Distr. join results
- Alternative join strategies
- Conclusions

Instrumentation

- ?

Instrumentation

- distributed EXPLAIN
  - (PLAN_TABLE @ master, + local ones)
- COLLECT COUNTERS
  - RSS stats, I/Os, buffer look-ups, comm. stats)
- FORCE OPTIMIZER

General measurements

Cost function:

– cpu + I/O + #msgs + #bytes

- w_{CPU} 0.0004 msec/instr (2.5MIPS)
- $w_{I/O}$ 23msec (actually: 17msec)
- w_{msg} 11msec (actually: 16msec)
- w_{byte} 0.002 msec/byte (4Mbits/sec)

Overview

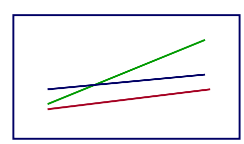
- Distr. computation and Optimization
- Instrumentation
- Distr. join results
- Alternative join strategies
- Conclusions

Distr. join results

- Q1: how to ship data - ship ‘whole’, or ‘fetch’, or?
- A1:

Distr. join results

- Q1: how to ship data - ship ‘whole’, or ‘fetch’, or?
- A1: ship outer to inner and back



Distr. join results

- Q2: Distr. vs local joins
- A2:

Distr. join results

- Q2: Distr. vs local joins
- A2: Distr. have more overhead, but better parallelism and buffer contention
- (distinguish: response time vs resource time)

CMU SCS

Distr. join results

- Q3: relative importance of cost components? (2.5MIPS, 20ms/IO, 11ms/msg, 4Mbps)
- A3:

CMU SCS

Distr. join results

- Q3: relative importance of cost components?
- A3: local cost is important
  - high speed net.: comm. cost < 10%
  - medium speed net: local cost still not negligible (50msec/msg, 40Kbps)

CMU SCS

Distr. join results

- Q4: Opt. evaluation - how often sub-optimal?
- A4: usually OK - problems with join selectivity est.
  - solution:?

CMU SCS

Distr. join results

- Q4: Opt. evaluation - how often sub-optimal?
- A4: usually OK - problems with join selectivity est.
  - solution: use estimate from previous execution!

CMU SCS

Overview

- Distr. computation and Optimization
- Instrumentation
- Distr. join results
- Alternative join strategies
- Conclusions

CMU SCS

Alternative join strategies

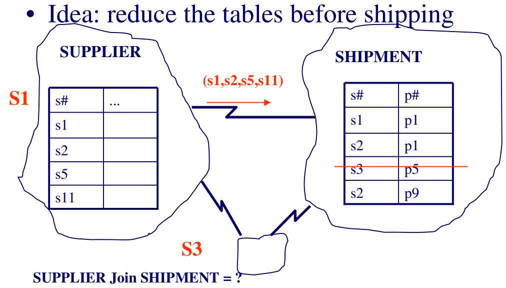
- 1) temporary indices
- 2) semi-joins
- 3) "Bloom"-joins

1) Temporary indices

- ship table T; build local index

2) Semijoins

- Idea: reduce the tables before shipping



SUPPLIER Join SHIPMENT = ?

Semijoins

- Idea: reduce the tables before shipping
- Eg., to reduce 'SHIPMENT'
  - send distinct values of SUPPLIER.S#

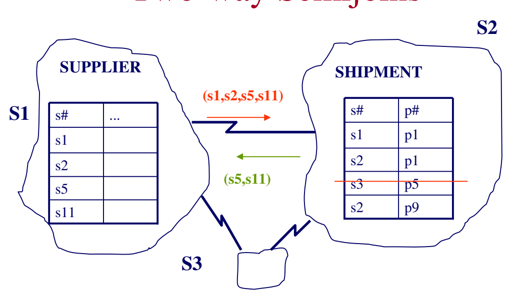
Semijoins

- Formally:
- SHIPMENT' = SHIPMENT $\ltimes$ SUPPLIER

A brilliant idea: two-way semijoins

- (not in book, not in final exam)
- reduce both relations with one more exchange: [Kang, '86]
- ship back the list of keys that didn't match

Two-way Semijoins



Two-way semijoins

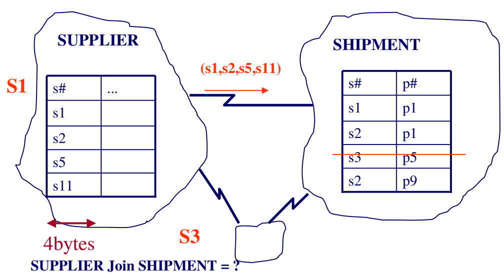
- ship back the list of keys that didn't match
- CAN NOT LOSE! (why?)
- further improvement:
  - or the list of ones that matched – whatever is shorter!

3) 'Bloom-joins'

- how to ship the projection, say, of SUPPLIER.s#, even cheaper?
- A: Bloom-filter [Lohman+] =
  - quick&dirty membership testing

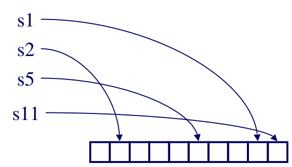
Bloom-join

- Idea: reduce table - using only,say, 10 bits? !!



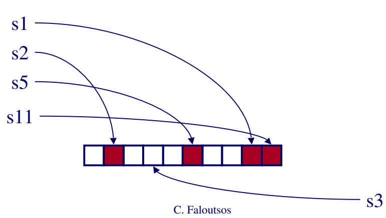
Bloom-join

- idea: use a bit-string and hashing
  - may have "false alarms" (OK!)



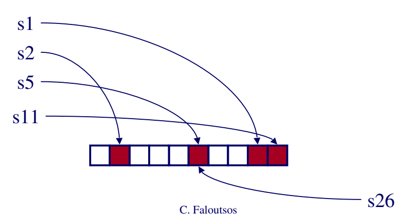
Bloom-join

- idea: use a bit-string and hashing
  - may have "false alarms" (OK!)



Bloom-join

- idea: use a bit-string and hashing
  - may have "false alarms" (OK!)



Bloom filters

- could set m>1 bits per value
- used for text retrieval (“Zato-coding”, in ‘49(!); signature files)
- differential files [Lohman+Severance]
- UNIX’s spell checker [McIlroy - IEEE COM’82]
- membership testing, in general

Bloom join

- Q1: How many false alarms, if we have
  - F bits, and
  - Ds (# of distinct values in table ‘S’)
  - Ct (# of tuples in table ‘T’)
  - SCt (# of tuples in ‘T’ that match)
- Q1’: How many ‘1’s, in the Bloom filter?
- A1’:

Bloom join

- Q1: How many false alarms, if we have
  - F bits, and
  - Ds (# of distinct values in table ‘S’)
  - Ct (# of tuples in table ‘T’)
  - SCt (# of tuples in ‘T’ that match)
- Q1’: How many ‘1’s, in the Bloom filter?
- A1’: $bits_S = F(1 - (1 - 1/F)^{Ds}) \sim F(1- exp(-Ds/F))$

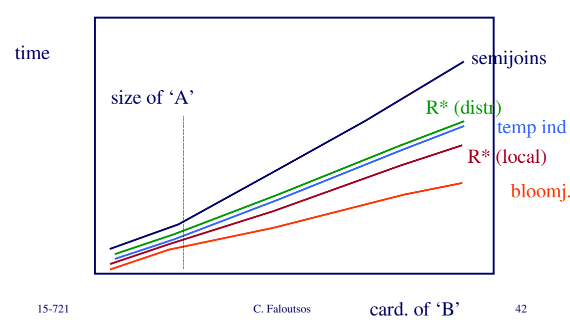
Bloom join

- Q1: How many false alarms, if we have
  - F bits, and
  - Ds (# of distinct values in table ‘S’)
  - Ct (# of tuples in table ‘T’)
  - SCt (# of tuples in ‘T’ that match)
- A1: $BCt = SCt + bits_S/F * (Ct - SCt)$
  - (book: slightly different formula)

Comparison

- A join B
- A: 1,000 tuples (query site)
- B: 100 - 6,000 tuples
- F=4Kb for bloom filter
- high/medium speed network
- R*, R*+temp-index, semijoin, bloom-join

Comparison



CMU SCS

Comparison

- Q: why bloom joins are better than semijoins?

CMU SCS

Comparison

- Q: why bloom joins are better than semijoins?
- A: lower **local** processing! (simple scan of 'B')
- (similar results for 'B' being the query site)

CMU SCS

Comparison - slower network

- bloom joins outperform all distributed algo's
  – (fewer bytes shipped)

CMU SCS

Conclusions

- ship whole inner table wins
- R* optimizer: accurate
- distribution of queries: often helps, due to
  – parallelism
  – more buffers
- local cost: not negligible