CMU - 15-415

Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture #27: Distributed DB
(R&G ch. 22)

General Overview

• Relational model - SQL
• Functional Dependencies & Normalization
• Physical Design; Indexing
• Query optimization
• Transaction processing
• Advanced topics
  – Spatial DB
  – Data Mining
  – Distributed Databases

Problem – definition

• centralized DB:

CHICAGO

LA

NY

C. Faloutsos
CMU SCS 15-415
Problem – definition

- Distr. DB:
- DB stored in many places
  - ... connected

LA

NY

EMP

EMPLOYEE

LA

NY

EMP

EMPLOYEE

LA

NY

EMP

EMPLOYEE

LA

NY

EMP

EMPLOYEE
Pros + Cons

• Pros
  – Data sharing
  – reliability & availability
  – speed up of query processing

• Cons
  – software development cost
  – more bugs
  – may increase processing overhead (msg)

Overview

• Problem – motivation
• Design issues
• Query optimization – semijoins
• transactions (recovery, conc. control)
Design of Distr. DBMS

what are our choices of storing a table?

• replication
• fragmentation (horizontal; vertical; hybrid)
• both
Transparency & autonomy

Issues/goals:
• naming and local autonomy
• replication and fragmentation transp.
• location transparency
  i.e.:

Problem – definition

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

Overview

• Problem – motivation
• Design issues
• Query optimization – semijoins
• transactions (recovery, conc. control)
Distributed Query processing

- issues (additional to centralized q-opt)
  - cost of transmission
    (cpu, disk, #bytes-transmitted, #messages-transmitted)
  - parallelism / overlap of delays
    minimize elapsed time?
    or minimize resource consumption?

Distr. Q-opt – semijoins

Semijoins

- choice of plans?
semijoins

- choice of plans?
- plan #1: ship SHIP -> S2; join; ship -> S3
- plan #2: ship SHIP->S3; ship SUP->S3; join
- ...
- others?

Distr. Q-opt – semijoins

Semijoins

- Idea: reduce the tables before shipping
Semijoins

- How to do the reduction, cheaply?
- Eg., reduce ‘SHIPMENT’:

Semijoins

- Idea: reduce the tables before shipping

Semijoins

- Formally:
  - $\text{SHIPMENT}' = \text{SHIPMENT} \bowtie \text{SUPPLIER}$
  - express semijoin w/ rel. algebra
Semijoins

- Formally:
  - \( \text{SHIPMENT}' = \text{SHIPMENT} \bowtie \text{SUPPLIER} \)
  - express semijoin w/ rel. algebra

\[
R' = R \bowtie S \\
= \pi_R(R \bowtie S)
\]

Semijoins – eg:

- suppose each attr. is 4 bytes
- Q: transmission cost (#bytes) for semijoin

\( \text{SHIPMENT}' = \text{SHIPMENT} \bowtie \text{SUPPLIER} \)

Semijoins

- Idea: reduce the tables before shipping

\[
\begin{array}{c|c|c|c|c|c|c}
S1 & x1 & x2 & x3 & x4 & x5 & x6 \\
S2 & s1 & s2 & s3 & s4 & s5 & s6 \\
S3 & 4 bytes & \text{SUPPLIER Join SHIPMENT} & \end{array}
\]

[Diagram showing the reduction of tables before shipping]
Semijoins – eg:

- suppose each attr. is 4 bytes
- Q: transmission cost (#bytes) for semijoin
  \[ \text{SHIPMENT}' = \text{SHIPMENT} \text{ semijoin SUPPLIER} \]
- A: 4*4 bytes

Semijoins – eg:

- suppose each attr. is 4 bytes
- Q1: give a plan, with semijoin(s)
- Q2: estimate its cost (#bytes shipped)

Semijoins – eg:

- A1:
  - reduce \text{SHIPMENT} to \text{SHIPMENT}'
  - \text{SHIPMENT}' \rightarrow S3
  - \text{SUPPLIER} \rightarrow S3
  - do join @ S3
- Q2: cost?
Semijoins

Semijoins – eg:

- **A2:**
  - 4*4 bytes - reduce SHIPMENT to SHIPMENT’
  - 3*8 bytes - SHIPMENT’ -> S3
  - 4*8 bytes - SUPPLIER -> S3
  - 0 bytes - do join @ S3

72 bytes TOTAL

Other plans?
Other plans?

P2:
• reduce SHIPMENT to SHIPMENT'
• reduce SUPPLIER to SUPPLIER'
• SHIPMENT' -> S3
• SUPPLIER' -> S3

Other plans?

P3:
• reduce SUPPLIER to SUPPLIER'
• SUPPLIER' -> S2
• do join @ S2
• ship results -> S3

A brilliant idea: ‘Bloom-joins’

• (not in the book – not in the final exam)
• how to ship the projection, say, of SUPPLIER.s#, even cheaper?
• A: Bloom-filter [Lohman+] =
  – quick&dirty membership testing
Another 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
• CAN NOT LOSE! (why?)
• further improvement:
  – or the list of ones that matched – whatever is shorter!
Overview

- Problem – motivation
- Design issues
- Query optimization – semijoins
- Transactions (recovery, conc. control)

Transactions – recovery

- Problem: eg., a transaction moves $100 from NY -> $50 to LA, $50 to Chicago
- 3 sub-transactions, on 3 systems, with 3 W.A.L.s
- How to guarantee atomicity (all-or-none)?
- Observation: additional types of failures (links, servers, delays, time-outs ...)

- Problem: eg., a transaction moves $100 from NY -> $50 to LA, $50 to Chicago
Distributed recovery

How?

CHICAGO

LA

NY

T1,1: -$100

T1,2: +$50

T1,3: +$50

NY

T1,1: -$100

T1,2: +$50

T1,3: +$50

Step 1: choose coordinator

Distributed recovery

• Step 2: execute a protocol, eg., “2 phase commit”
2 phase commit

T1,1 (coord.)
T2
T1,3

prepare to commit

Y
Y

commit

CMU 15-415
C. Faloutsos
2 phase commit (eg., failure)

T1,1 (coord.)  T1,2  T1,3

prepare to commit

time

2 phase commit

T1,1 (coord.)  T1,2  T1,3

prepare to commit

Y  N

time

2 phase commit

T1,1 (coord.)  T1,2  T1,3

prepare to commit

Y  N

abort

time
Distributed recovery

- Many, many additional details (what if the coordinator fails? what if a link fails? etc)
- and many other solutions (eg., 3-phase commit)

Overview

- Problem – motivation
- Design issues
- Query optimization – semijoins
- transactions (recovery, conc. control)

Distributed conc. control

- also more complicated:
- distributed deadlocks!
Distributed deadlocks

LA

NY
Distributed deadlocks

- cites need to exchange wait-for graphs
- clever algorithms, to reduce # messages

Conclusions

- Distr. DBMSs: not deployed
- BUT: produced clever ideas:
  - semijoins
  - distributed recovery / conc. control
- which can be useful for
  - parallel db / clusters
  - ‘active disks’
  - replicated db (e-commerce servers)