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:
Problem – definition

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

LA

NY

EMPLOYEE

EMP

connect to LA; exec sql select * from EMP; ...
connect to NY; exec sql select * from EMPLOYEE; ...

now:

DBMS1

DBMS2

D-DBMS

D-DBMS

Problem – definition

LA

NY

EMPLOYEE

EMP

connect to distr-LA; exec sql select * from EMPLOYEE; ...

ideally:

DBMS1

DBMS2

D-DBMS

D-DBMS
Pros + Cons

• Pros
  –
  –
  –
  –
• Cons
  –
  –
  –
  –

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

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>wall str.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>234</td>
<td>johnson</td>
<td>sunset blvd</td>
</tr>
</tbody>
</table>
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

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

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
Semijoins – eg:

• suppose each attr. is 4 bytes
• Q: transmission cost (#bytes) for semijoin
  SHIPMENT’ = SHIPMENT 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 SHIPMENT to SHIPMENT’
  – SHIPMENT’ -> S3
  – SUPPLIER -> 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 ....)
Distributed recovery

How?

CHICAGO
T1,2: +$50
T1,1:$100

LA
T1,3: +$50

NY

Distributed recovery

Step 1: choose coordinator

CHICAGO
T1,2: +$50
T1,1:$100

LA
T1,3: +$50

NY

Distributed recovery

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

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

prepare to commit

Y

Y

commit
2 phase commit (eg., failure)

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

prepare to commit

Y
N

abort

2 phase commit

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

prepare to commit

Y
N
Distributed recovery

- Many, many additional details (what if the coordinator fails? what if a link fails? etc)
- and many other solutions (e.g., 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

NY

LA

T_{1,la}

T_{2,la}

T_{1,ny}

T_{2,ny}

CHICAGO

Distributed deadlocks

NY

LA

T_{1,la}

T_{2,la}

T_{1,ny}

T_{2,ny}

Distributed deadlocks

NY

LA

T_{1,la}

T_{2,la}

T_{1,ny}

T_{2,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)