**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

---

**Problem – definition**

- centralized DB:
  - CHICAGO

**Problem – definition**

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

---

**Problem – definition**

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

**Problem – definition**

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

- ?

What’s new?

- Q-opt

What’s new?

- Q-opt
  - communication cost
  - larger search space
  - load balance
  - speed, cost, space, time differences on machines

- CC
  - need distributed algorithms (distr. deadlock)
What’s new?

- Q-opt
- CC
- Recovery

What’s new?

- Q-opt
- CC
- Recovery
  - much more complex; more parts that can fail;
  - ‘2 phase commit’

What’s new?

- Q-opt
- CC
- Recovery
- multiple copies; fragments
- multiple copies; fragments
  - (but, at most 2, in practice)

D-DBMS in practice

Why would one need a D-DBMS?

- geographic distribution / performance
- off-loading mainframes with local processing
- ‘sins of the past’ - integrating legacy systems
D-DBMS in practice

• there are products (IBM Data Joiner, Oracle*)
• BUT: they are not commercially as successful as we would expect! - why?

D-DBMS - why not?

Speculations:
• data warehouses (copy DBs locally! Sears, Wal-Mart, Kmart)
• D-DBMSs would cut down sales of D/W products
• distr. q-opt is immature

D-DBMS - other issues?

Integration of data sources: desirable, because of the web - remaining issues:
• semantic consistency (e.g., salaries before/after taxes)
• authentication
• 2-phase-commit on top of legacy databases

Conclusions

D-DBMS research produced great ideas, useful for
• parallel dbms / “active disks” / sensors
• p2p (peer to peer networks)
• ’middle-ware’

Conclusions

Namely:
• 2 phase commit
• distributed q-opt - semi-joins/bloom-joins
• distributed catalogue
• distributed deadlock detection

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
System R* architecture

Citation

Detailed outline

• Environment
• Object naming
• Distributed Catalogues
• Xact management - commit protocols
• Q-opt
• CC- recovery
• SQL changes

Environment

LA

NY

Environment

LA

NY

Object naming

No global naming system (why?)
Instead: System Wide Names (SWN)
• by attaching 'site' on user-names
• by attaching 'birth-site' on tables
  e.g.:
  bruce.EMPLOYEE ->
  bruce@san-jose . EMPLOYEE@yorktown
Distributed catalogues

• Q: where and how should we store the schema?
• A1: fully replicated (but:....)
• A2: single copy (but:....)
• A3?

Distributed catalogues

• Q: where and how should we store the schema?
• A1: fully replicated (but:....)
• A2: single copy (but:....)
• A3: only birth site keeps moving info - thus

Distributed catalogues

• A3: only birth site keeps moving info - thus each site has
  – local schema +
  – moving info (for items ‘born’ here) and
  – birth sites of global objects
• thus: <= 2 messages are enough to locate non-local object

Detailed outline

• Environment
• Object naming
• Distributed Catalogues
• Xact management - commit protocols
• Q-opt
• Query execution
• SQL changes

Xact management

• Q: how to give xact-ids?
• A: site-id & sequence#
  – ordered (to break deadlocks)
Transactions – recovery

- Problem: e.g., 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

- Step 1: choose coordinator
- Step 2: execute a protocol, eg., “2 phase commit”

2 phase commit

- T1,1 (coord.) prepare to commit
- T1,2
- T1,3
- 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)

Detailed outline

- Environment
- Object naming
- Distributed Catalogues
- Xact management - commit protocols
  - Q-opt
  - Query execution
  - SQL changes

Distributed Q-opt

- Steps:
  - parse
  - resolve names
  - authorization
  - compilation + plan generation
    - binding? (eg., an access path may be dropped mid-flight!)

Distributed Q-opt

- Q: how to do binding?
  - A1: at a chosen site (-> ~ centralized)
  - A2: at the originating site (but: needs much info, which may be out-dated)
  - A3: distributed binding

Distributed binding

- master site decides inter-site issues + high level binding
- local sites do low-level decisions

Local optimality: OK
Global optimality: NOK
Solution: Master sites send global plan; local sites complain, if things changed

Distributed q-opt

- cost to minimize?
Distributed q-opt

- cost to minimize?
- cost = CPU + I/O + communication
  - comm. cost = 
    - msg-cost * #messages + 
    - byte-cost * #bytes
- (could minimize elapsed time, instead...)

Distr. q-opt - join plans

Joins: join order + join method + LOCATION

Distr. q-opt - join plans

SEVERAL choices - R* chooses one of 5:
- (a) ship inner to S1: join there
- (b) ship outer to S2, tuple-at-a-time
- (c) (`semi-join`): reduce inner; ship that to S1
- (d) ship both tables to a third site
- (e) ship outer to a third site; do (c)

Semijoins

- Idea: reduce the tables before shipping

Semijoins

- Formally:
  - $\text{SHIPMENT}^* = \text{SHIPMENT} \bowtie \text{SUPPLIER}$
Detailed outline

- Environment
- Object naming
- Distributed Catalogues
- Xact management - commit protocols
- Q-opt
- Query execution
- SQL changes

Distributed deadlocks

- LA
- NY

Distributed deadlocks

- T_{1La}
- T_{1Lm}
- T_{2La}
- T_{2Lm}

- LA
- NY

Distributed deadlocks

- cites need to exchange wait-for graphs
- clever algorithms, to reduce # messages
  - naively: each site ships its wait-for strings, until all have all
  - anything better?
### Distributed deadlocks

- anything better?
- A: each site ships ONLY the strings where ‘first-xact-id’ < ‘last-xact-id’
  - (any other ordering, is fine!)
- Eg: LA: T1 -> T2; NY T2 -> T1
  - only NY will send

### SQL extensions

- DEFINE SYNONYM <rel-name> AS <SWN>
- DISTRIBUTE TABLE <t-name> HORIZONTALLY | VERTICALLY | REPLICATED ...
- DEFINE SNAPSHOT ...
- REFRESH SNAPSHOT
- MIGRATE TABLE ...

### Conclusions

- 2 phase commit
- distributed q-opt; distr. deadlock detection
- distributed catalogue

### Detailed outline

- Environment
- Object naming
- Distributed Catalogues
- Xact management - commit protocols
- Q-opt
- Query execution
- SQL changes