15-721 DB Sys. Design & Impl.

Distributed DBMSs & R*

Christos Faloutsos

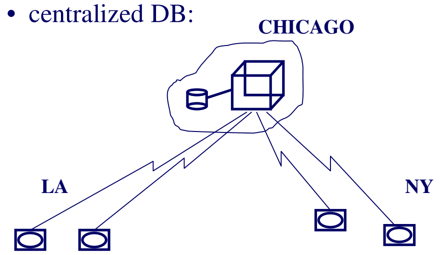
www.cs.cmu.edu/~christos

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

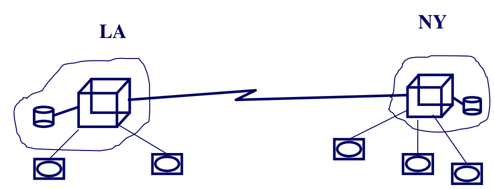
Problem – definition

- centralized DB:



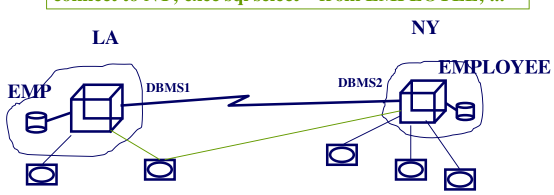
Problem – definition

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



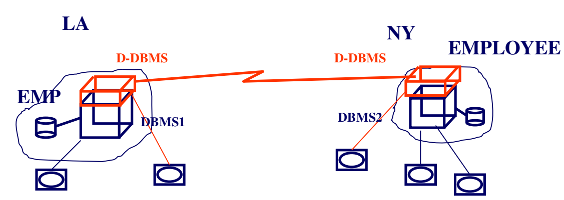
Problem – definition

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

- ?

Requirements

- location transparency
- performance transparency (-> distr. q-opt)
- copy transparency
- transactions transparency
- fragment transparency
- schema transparency
- local dbms transparency
- (no system has all these features)

What's new?

- Q-opt

What's new?

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

What's new?

- Q-opt
- CC

What's new?

- Q-opt
- CC
  - need distributed algorithms (distr. deadlock)

CMU SCS

What's new?

- Q-opt
- CC
- Recovery

15-721 C. Faloutsos 13

CMU SCS

What's new?

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

15-721 C. Faloutsos 14

CMU SCS

What's new?

- Q-opt
- CC
- Recovery
- multiple copies; fragments

15-721 C. Faloutsos 15

CMU SCS

What's new?

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

15-721 C. Faloutsos 16

CMU SCS

D-DBMS in practice

Why would one need a D-DBMS?

15-721 C. Faloutsos 17

CMU SCS

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

15-721 C. Faloutsos 18

CMU SCS

D-DBMS in practice

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

15-721 C. Faloutsos 19

CMU SCS

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

15-721 C. Faloutsos 20

CMU SCS

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

15-721 C. Faloutsos 21

CMU SCS

Conclusions

D-DBMS research produced great ideas, useful for

- parallel dbms / "active disks" / sensors
- p2p (peer to peer networks)
- 'middle-ware'

15-721 C. Faloutsos 22

CMU SCS

Conclusions

Namely:

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

15-721 C. Faloutsos 23

CMU SCS

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

15-721 C. Faloutsos 24

System R* architecture

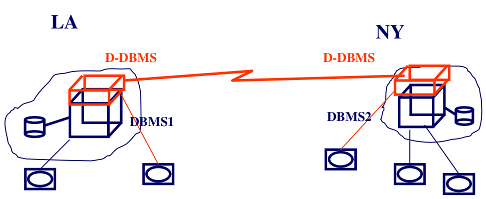
Citation

R. Williams, et al., "*R* : An Overview of the Architecture*." IBM Research Report RJ3325

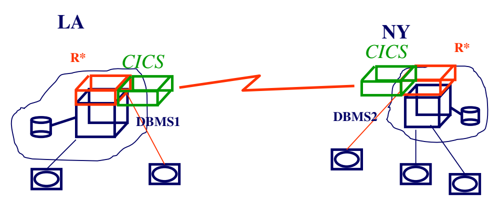
Detailed outline

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

Environment



Environment



Environment

Relations:

- dispersed;
- replicated
- fragmented (hor./vert.);
- snapshots

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

CMU SCS

Distributed catalogues

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

15-721 C. Faloutsos 31

CMU SCS

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

15-721 C. Faloutsos 32

CMU SCS

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

15-721 C. Faloutsos 33

CMU SCS

Detailed outline

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

15-721 C. Faloutsos 34

CMU SCS

Xact management

- Q: how to give xact-ids?

15-721 C. Faloutsos 35

CMU SCS

Xact management

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

15-721 C. Faloutsos 36

Transactions – recovery

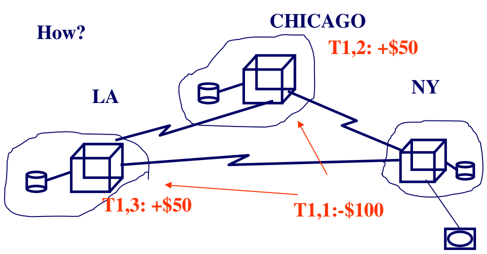
- Problem: eg., a transaction moves
  $100 from NY -> $50 to LA, $50 to Chicago
- 3 sub-transactions, on 3 systems, with 3 W.A.L.s
- how to guarantee atomicity (all-or-none)?
- Observation: additional types of failures (links, servers, delays, time-outs ....)

Transactions – recovery

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

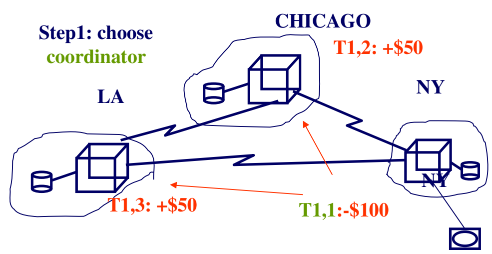
Distributed recovery

How?



Distributed recovery

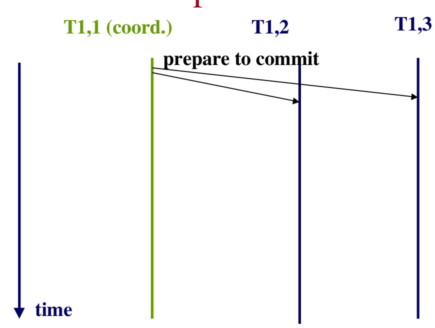
Step1: choose coordinator



Distributed recovery

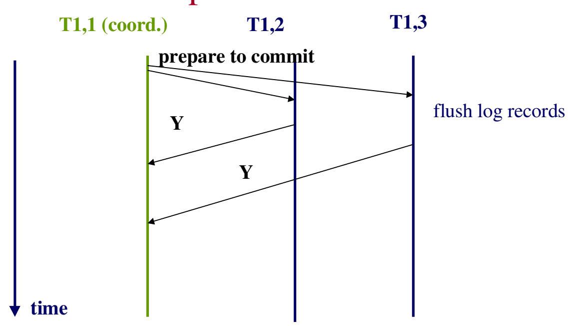
- Step 2: execute a protocol, eg., “2 phase commit”

2 phase commit



CMU SCS

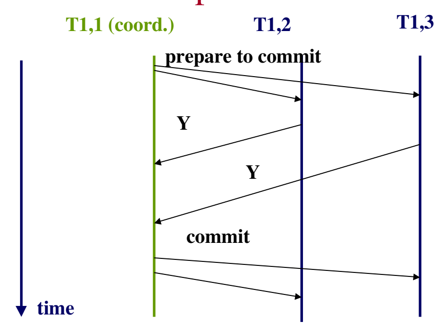
2 phase commit



15-721 C. Faloutsos 43

CMU SCS

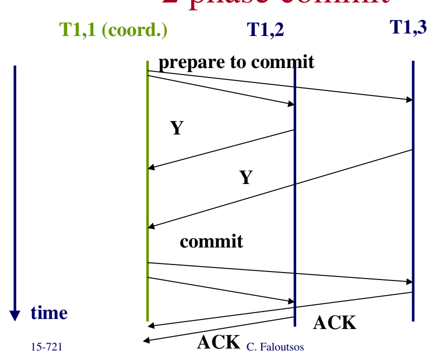
2 phase commit



15-721 C. Faloutsos 44

CMU SCS

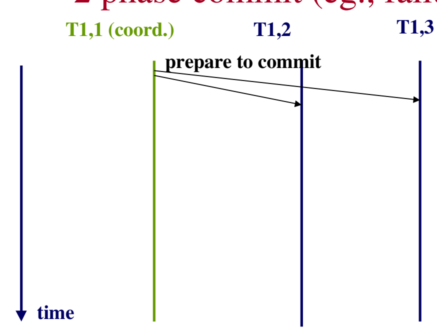
2 phase commit



15-721 C. Faloutsos 45

CMU SCS

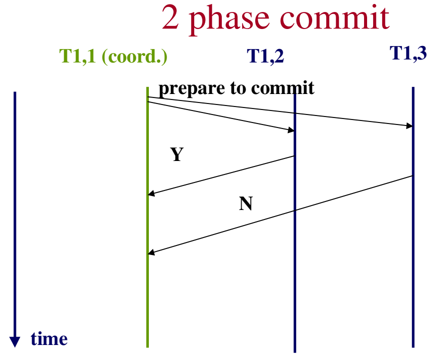
2 phase commit (eg., failure)



15-721 C. Faloutsos 46

CMU SCS

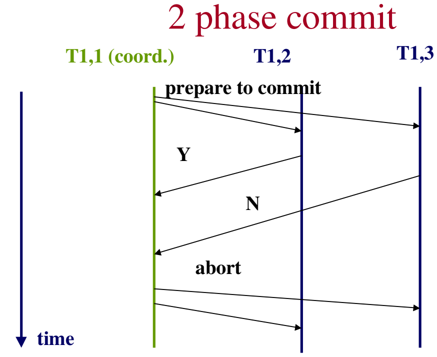
2 phase commit



15-721 C. Faloutsos 47

CMU SCS

2 phase commit



15-721 C. Faloutsos 48

CMU SCS

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)

CMU SCS

Detailed outline

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

CMU SCS

Distributed Q-opt

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

CMU SCS

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

CMU SCS

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 sends global plan; local sites complain, if things changed

CMU SCS

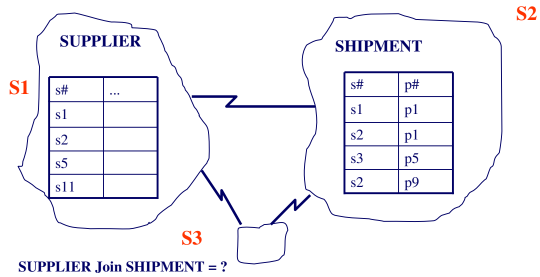
Distributed q-opt

- 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 –joins



Distr. q-opt - join plans

Joins: join order + join method + <u>LOCATION</u>

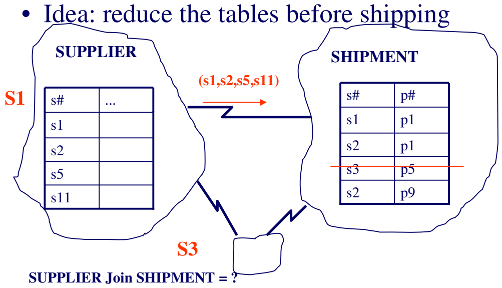
Distr. q-opt - join plans

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:
- SHIPMENT’ = SHIPMENT $\ltimes$ SUPPLIER

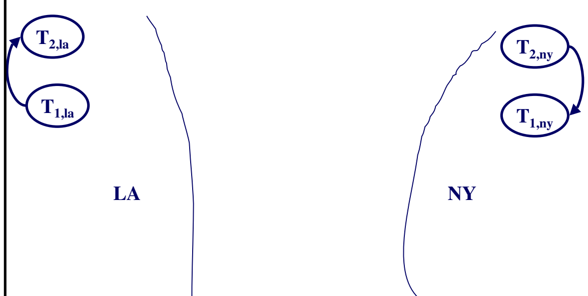
Detailed outline

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

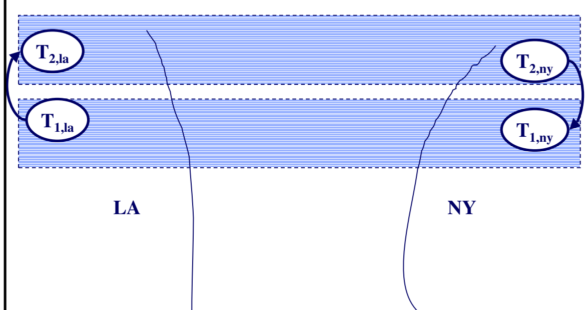
Detailed outline

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

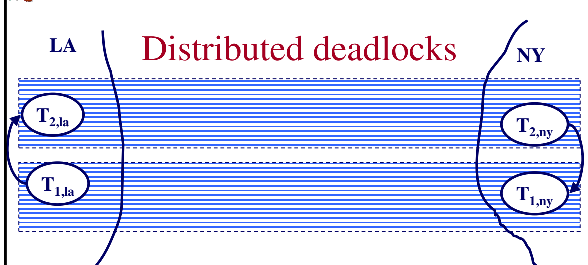
Distributed deadlocks



Distributed deadlocks



Distributed deadlocks



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

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

Detailed outline

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

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