Distributed Databases

CSE 532, Theory of Database Systems

Stony Brook University

http://www.cs.stonybrook.edu/~cse532

What is a Distributed Database?

- Database whose relations *reside* on different sites
- Database some of whose relations are *replicated* at different sites
- Database whose relations are *split* between different sites

Two Types of Applications that Access Distributed Databases

- The application accesses data at the level of SQL statements
  - *Example*: company has nationwide network of warehouses, each with its own database; a transaction can access all databases using their schemas
- The application accesses data at a database using only stored procedures provided by that database.
  - *Example*: purchase transaction involving a merchant and a credit card company, each providing stored subroutines for its subtransactions

Optimizing Distributed Queries

- Only applications of the first type can access data directly and hence employ query optimization strategies
- These are the applications we consider in this chapter

Some Issues

- How should a distributed database be designed?
- At what site should each item be stored?
- Which items should be replicated and at which sites?
- How should queries that access multiple databases be processed?
- How do issues of query optimization affect query design?

Why Might Data Be Distributed

- Data might be distributed to minimize communication costs or response time
- Data might be kept at the site where it was created so that its creators can maintain control and security
- Data might be replicated to increase its availability in the event of failure or to decrease response time

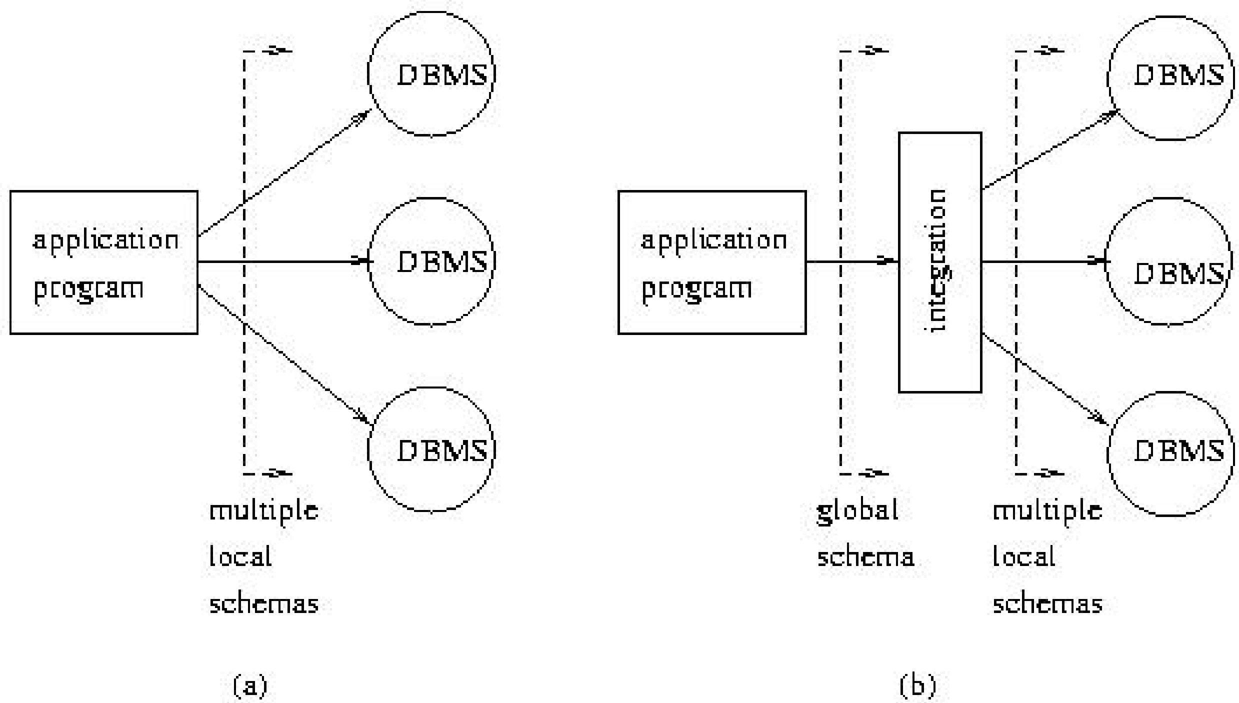
Application Designer's View of a Distributed Database

- Designer might see the individual schemas of each local database -- called a ***multidatabase*** **--** in which case distribution is visible
  - Can be ***homogeneous*** (all databases from one vendor) or ***heterogeneous*** (databases from different vendors)
- Designer might see a single ***global schema*** that integrates all local schemas (is a view) in which case distribution is hidden
- Designer might see a ***restricted global schema***, which is the union of all the local schemas
  - Supported by some vendors of homogeneous systems

Views of Distributed Data

(a) Multidatabase with local schemas

(b) Integrated distributed database with global schema



Multidatabases

- Application must explicitly connect to each site
- Application accesses data at a site using SQL statements based on that site's schema
- Application may have to do reformatting in order to integrate data from different sites
- Application must manage replication
  - Know where replicas are stored and decide which replica to access

Global and Restricted Global Schemas

- Middleware provides integration of local schemas into a global schema
  - Application need not connect to each site
  - Application accesses data using global schema
    - Need not know where data is stored – ***location transparency***
  - Global joins are supported
  - Middleware performs necessary data reformatting
  - Middleware manages replication – ***replication transparency***

Partitioning

- Data can be distributed by storing individual tables at different sites
- Data can also be distributed by decomposing a table and storing portions at different sites – called ***partitioning***
- Partitioning can be ***horizontal*** or ***vertical***

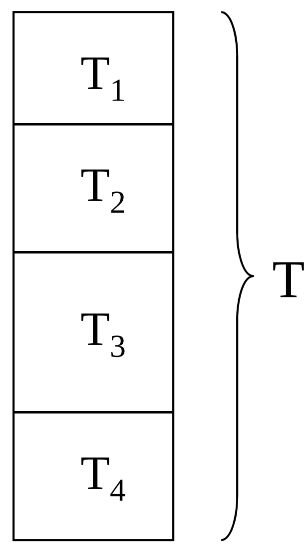
Horizontal Partitioning

- Each partition, T_i , of table T contains a subset of the rows and each row is in exactly one partition:

$$T_i = \sigma_{C_i}(T)$$

$$T = \bigcup T_i$$

  - Horizontal partitioning is lossless



Horizontal Partitioning

- *Example*: An Internet grocer has a relation describing inventory at each warehouse

  Inventory(*StockNum*, *Amount*, *Price*, *Location*)

- It partitions the relation by location and stores each partition locally: rows with *Location* = 'Chicago' are stored in the Chicago warehouse in a partition

  Inventory_ch(*StockNum*, *Amount*, *Price*, *Location*)

- Alternatively, it can use the schema

  Inventory_ch(*StockNum*, *Amount*, *Price*)

Vertical Partitioning

- Each partition, T_i, of T contains a subset of the columns, each column is in at least one partition, and each partition includes the key:

  $$T_i = \pi_{attr\_list_i}(T)$$
  $$T = T_1 \bowtie T_2 \ldots\ldots \bowtie T_n$$

  - Vertical partitioning is lossless
- *Example*: The Internet grocer has a relation

  Employee(*SSnum*, *Name*, *Salary*, *Title*, *Location*)

  - It partitions the relation to put some information at headquarters and some elsewhere:

  Emp1(*SSnum*, *Name*, *Salary*) – at headquarters

  Emp2(*SSnum*, *Name*, *Title*, *Location*) – elsewhere

Replication

- One of the most useful mechanisms in distributed databases
- Increases
  - Availability
    - If one replica site is down, data can be accessed from another site
  - Performance:
    - Queries can be executed more efficiently because they can access a local or nearby copy
    - Updates might be slower because all replicas must be updated

Replication Example

- Internet grocer might have relation

  Customer(*CustNum*, *Address*, *Location*)

  - Queries are executed
    - At headquarters to produce monthly mailings
    - At a warehouse to obtain information about deliveries
  - Updates are executed
    - At headquarters when new customer registers and when information about a customer changes

Example (con't)

- Intuitively it seems appropriate to *either* or *both*:
  - Store complete relation at headquarters
  - Horizontally partition a replica of the relation and store a partition at the corresponding warehouse site
- Each row is replicated: one copy at headquarters, one copy at a warehouse
- The relation can be both distributed *and* replicated

Example (con't): Performance Analysis

- We consider three alternatives:
  - Store the entire relation at the headquarters site and nothing at the warehouses (no replication)
  - Store the partitions at the warehouses and nothing at the headquarters (no replication)
  - Store entire relation at headquarters and a partition at each warehouse (replication)

Example (con't): Performance Analysis - Assumptions

- To evaluate the alternatives, we estimate the amount of information that must be sent between sites.
- Assumptions:
  - The Customer relation has 100,000 rows
  - The headquarters mailing application sends each customer 1 mailing a month
  - 500 deliveries are made each day; a single row is read for each delivery
  - 100 new customers/day
  - Changes to customer information occur infrequently

Example: The Evaluation

- Entire relation at headquarters, nothing at warehouses
  - 500 tuples per day from headquarters to warehouses for deliveries
- Partitions at warehouses, nothing at headquarters
  - 100,000 tuples per month from warehouses to headquarters for mailings (3,300 tuples per day, amortized)
  - 100 tuples per day from headquarters to warehouses for new customer registration
- Entire relation at headquarters, partitions at warehouses
  - 100 tuples per day from headquarters to warehouses for new customer registration

Example: Conclusion

- Replication (case 3) seems best, if we count the number of transmissions.
- Let us look at other measures:
  - If no data stored at warehouses, the time to handle deliveries might suffer because of the remote access (probably not important)
  - If no data is stored at headquarters, the monthly mailing requires that 100,000 rows be transmitted in a single day, which might clog the network
  - If we replicate, the time to register a new customer might suffer because of the remote update
    - But this update can be done by a separate transaction after the registration transaction commits (***asynchronous update***)

Query Planning

- Systems that support a global schema contain a global query optimizer, which analyzes each global query and translates it into an appropriate sequence of steps to be executed at each site
- In a multidatabase system, the query designer must manually decompose each global query into a sequence of SQL statements to be executed at each site
  - Thus a query designer must be her own query optimizer

Global Query Optimization

- A familiarity with algorithms for global query optimization helps the application programmer in designing
  - Global queries that will execute efficiently for a particular distribution of data
  - Algorithms for efficiently evaluating global queries in a multidatabase system
  - The distribution of data that will be accessed by global queries

Planning Global Joins

- Suppose an application at site A wants to join tables at sites B and C. Two straightforward approaches
  - Transmit both tables to site A and do the join there
    - The application explicitly tests the join condition
    - This approach must be used in multidatabase systems
  - Transmit the smaller of the tables, e.g. the table at site B, to site C; execute the join there; transmit the result to site A
    - This approach might be used in a homogenous distributed database system

Global Join Example

- Site B

  Student(*Id*, *Major*)

- Site C

  Transcript(*StudId*, *CrsCode*)

- Application at Site A wants to compute join with join condition

  Student.*Id* = Transcript.*StudId*

Assumptions

- Lengths of attributes
  - *Id* and *StudId*: 9 bytes
  - *Major*: 3 bytes
  - *CrsCode*: 6 bytes
- Student: 15,000 tuples, each of length 12 bytes
- Transcript: 20,000 tuples, each of length 15 bytes
  - 5000 students are registered for at least 1 course (10,000 students are not registered – summer session)
  - Each student is registered for 4 courses on the average

Comparison of Alternatives

- Send both tables to site A, do join there:
  - have to send 15,000*12 + 20,000*15 = 480,000 bytes
- Send the smaller table, Student, from site B to site C, compute the join there. Then send result to Site A:
  - have to send 15,000*12 + 20,000*18 = 540,000 bytes

- Alternative 1 is better

Another Alternative: Semijoin

- Step1:

  At site C: Compute $P = \pi_{StudId}(\text{Transcript})$

  Send P to site B

  - P contains Ids of students registered for at least 1 course
  - Student tuples having Ids not in P do not contribute to join, so no need to send them
- Step 2:

  At site B: Compute $Q = \text{Student} \rtimes_{Id\,=\,StudId} P$

  Send Q, to site A

  - Q contains tuples of Student corresponding to students registered for at least 1 course (i.e., 5,000 students out of 15,000)
  - Q is a *semijoin* – the set of all Student tuples that will participate in the join
- Step 3:

  Send Transcript to site A

  At site A: Compute $\text{Transcript} \bowtie_{Id\,=\,StudId} Q$

Comparison Semijoin with Previous Alternatives

- In step 1: 45,000 = 5,000*9 bytes sent
- In step 2: 60,000 = 5,000*12 bytes sent
- In step 3: 300,000 = 20,000*15 bytes sent
- In total: 405,000 = 45,000 + 60,000 + 300,000 bytes sent

- Semijoin is the best of the three alternatives

Definition of Semijoin

- The ***semijoin*** of two relations, T_1 and T_2, is defined as:

$$T_1 \ltimes_{join\_cond} T_2 = \pi_{attributes(T_1)}(T_1 \bowtie_{join\_cond} T_2)$$
$$= T_1 \bowtie \pi_{join\text{-}attributes}(T_2)$$

  - In other words, the semijoin consists of the tuples in T_1 that participate in the join with T_2

Using the Semijoin

- To compute $T_1 \bowtie_{join\_cond} T_2$ using a semijoin, first compute $T_1 \ltimes_{join\_cond} T_2$ then join it with T_2:

$$\pi_{attributes(T_1)}(T_1 \bowtie_{join\_cond} T_2) \bowtie_{join\_cond} T_2$$

Queries that Involve Joins and Selections

- Suppose the Internet grocer relation Employee is vertically partitioned as

  Emp1(*SSnum*, *Name*, *Salary*) at Site B

  Emp2(*SSnum*, *Title*, *Location*) at Site C
- A query at site A wants the names of all employees with *Title* = 'manager' and *Salary* > '20000'
- **Solution 1:** First do join then selection:

$$\pi_{Name}(\sigma_{Title=\text{'manager'} \text{ AND } Salary>\text{'20000'}}(\text{Emp1} \bowtie \text{Emp2}))$$

  - Semijoin *not* helpful here: all tuples of each table must be brought together to form the join (the join is on SSNum)

Queries that Involve Joins and Selections

- **Solution 2**: Do selections before the join:

  $\pi_{Name}((\sigma_{Salary > '20000'}(Emp1)) \bowtie (\sigma_{Title = 'manager'}(Emp2)))$

- At site B, select all tuples from Emp1 satisfying *Salary* > '20000'; call the result R1
- At site C, select all tuples from Emp2 satisfying *Title*='manager'; call the result R2
- At some site to be determined by minimizing communication costs, compute $\pi_{Name}(R1 \bowtie R2)$;

  Send result to site A
  - In a multidatabase, join must be performed at Site A, but communication costs are reduced because only "selected" data needs to be sent

Summary: Choices to be Made by a Distributed Database Application Designer

- Place tables at different sites
- Partition tables in different ways and place partitions at different sites
- Replicate tables or data within tables and place replicas at different sites
- In multidatabase systems, do manual "query optimization": choose an optimal sequence of SQL statements to be executed at each site