### Overview of Distributed Databases

- **Topics**
  - Concepts
  - Benefits & Problems
  - Architectures

- **Readings**
  - Lecture Notes
  - Oszu & Valduriez
    - Selected sections of chapters 1 & 4

### Why distributed

- Corresponds to the organizational structure of distributed enterprises
  - Electronic commerce
  - Manufacturing control systems
- Divide and Conquer
  - Work towards a common goal
- Economic Benefits
  - More computing power
  - More discipline in smaller but autonomous groups

### Why distributed

To solve complex problems by dividing into smaller pieces, and assigning them to different software groups, which work on different computers

To produce a system which, although runs on multiple processing elements, can work effectively towards a common goal

### Distributed Database Systems

Union of two opposed technologies

- **Centralize**
- **Distribute**

### Concepts

- Distributed Computing Systems
  - A number of autonomous processing elements, not necessarily homogeneous, that are interconnected by a computer network and that cooperate in performing their assigned tasks
- Distributed Database
  - A collection of multiple, logically interrelated databases, distributed over a computer network
- Distributed Database Management System
  - A software that manages a distributed database, while making the distribution transparent to the user

### What is being distributed?

- Processing logic
  - Inventory
  - Personnel
  - Sales
- Function
  - Printing
  - Email
- Data

---

Lecture Notes INFS7907

© School of Information Technology and Electrical Engineering
**Misconceptions**

Distributed Computing is not...

- Multi-processor system
- Backend processor
- Computer network
- Parallel Computing

---

**Centralized Database**

![Centralized Database Diagram]

**Networked Architecture**

![Networked Architecture Diagram]

**Distributed database architecture**

![Distributed database architecture Diagram]

---

**Benefits of DDBs**

- Transparency
- Reliability
- Performance
- Expansion

---

**Transparency**

- Separation of high-level semantics from low-level implementation issues
- Extension of the Data Independence concept in centralized databases
- Basic Concepts
  - Fragmentation
  - Replication
**Transparency Types in DDBMSs**

- Network
- Replication
- Fragmentation

**Network Transparency**
- Protect the user from the operational details of the network.
- Users do not have to specify where the data is located
  - Location Transparency
  - Naming Transparency

**Replication Transparency**
- Replicas (copies of data) are created for performance and reliability reasons
- Users should be made unaware of the existence of these copies
- Replication causes update problems

**Fragmentation Transparency**
- Fragments are parts of a relation
  - Vertical Fragments: Subset of Columns
  - Horizontal Fragment: Subset of tuples
- Fragments are also created for performance and reliability reasons
- Users should be made unaware of the existence of fragments

**Reliability**
- Replicated components (data & software) eliminate single points of failure
- Failure of a single site or communication link will not bring down entire system
- Managing the ‘reachable’ data requires Distributed Transaction Support which provides correctness in the presence of failures

**Performance**
- Data Localization
  - Reduces contention for CPU and I/O services
  - Reduces communication overhead
- Parallelism
  - Inter-query & Intra-query parallelism
  - The ‘Multiplex’ approach
**Expansion**

- Expansion by adding processing and storage power to the network
- Replacing a Mainframe vs. Adding more micro-computers to the network

**Problem Areas**

- Database Design
- Query Processing
- Directory Management
- Concurrency Control
- Deadlock Management
- Reliability issues
- Operating System Support
- Heterogeneous Databases

**Database Design**

- How to place database and applications
- Fragmentation or Replication
  - Disjoint Fragments
  - Full Replication
  - Partial Replication
- Design Issues
  - Fragmentation
  - Allocation

**Query Processing**

- Determining Factors
  - Distribution of Data
  - Communication Costs
  - Lack of locally available data
- Objective
  - Maximum utilisation of inherent parallelism

**Directory Management**

- Contains information on data descriptions plus locations
- Problems similar to database design
  - global or local
  - centralised or distributed
  - single copy or multiple copies

**Concurrency Control**

- Integrity of the database as in centralised plus consistency of multiple copies of the database - Mutual Consistency
- Approaches
  - Pessimistic: Synchronising before
  - Optimistic: Checking after
- Strategies
  - Locking
  - Time stamping
**Deadlock Management**

- Similar solutions as for centralised
  - Prevention
  - Avoidance
  - Detection
  - Recovery

**Reliability issues**

- Reliability is an advantage, but does not come automatically
- On failure of a site
  - Database at operational sites must remain consistent and up to date
  - Database at failed site must recover after failure

**Operating System Support**

- Single processor systems
  - Memory management
  - File system and access methods
  - Crash recovery
  - Process management
- Distributed environments (in addition to above)
  - Dealing with multiple layers of network software

**Heterogeneous Databases**

- Databases differ in
  - Data Model
  - Data Language
- Translation mechanism for data and programs
- Multi database systems
  - Building a distributed DBMS from a number of autonomous, centralised databases

**Architectural Models**

- Architecture defines structure
  - Components
  - Functionality of Components
  - Interactions and Interrelationships

The ANSI/SPARC Architecture for Centralised DBMS

**Classification**

1. Autonomy
2. Distribution
3. Heterogeneity
Autonomy
- Distribution of control (not data)
  - Design
  - Communication
  - Execution
- Alternatives
  - Tight integration
  - Semi-autonomous
  - Total isolation / Fully Autonomous

Distribution
- Distribution of data
- Physical distribution over multiple sites
- Alternatives
  - Non-distributed / Centralised
  - Client Server
  - Peer-to-peer / Fully distributed

Heterogeneity
- Different database systems
- Major differences
  - Data models (Relational, Hierarchical)
  - Data languages (SQL, QBE)
  - Transaction Management Protocols

Architecture Alternatives
- Autonomy (A)
  - A0: Tight integration
  - A1: Semi-autonomous
  - A2: Total isolation
- Distribution (D)
  - D0: Non-distributed
  - D1: Client Server
  - D2: Peer-to-peer
- Heterogeneity (H)
  - H0: Homogeneous
  - H1: Heterogeneous

Some alternatives are meaningless or not practical!

Architectural Alternatives
- (A0, D0, H0)
  - A collection of logically integrated DBMSs on the same site, also called Composite Systems
- (A0, D0, H1)
  - Providing integrated access to heterogeneous systems on a single machine
- (A0, D1, H0)
  - Client Server distribution
- (A0, D2, H0)
  - Fully distributed
- (A1, D0, H0)
  - Semi-autonomous systems, also called Federated Systems. Each DBMS knows how to participate in the federation
**Architectural Alternatives**

- (A1, D0, H1)
  - Heterogeneous Federated DBMSs.
- (A1, D1, H1)
  - Distributed Heterogeneous Federated DBMSs
- (A2, D0, H0)
  - Multi-database Systems. Complete homogeneity in component systems is unlikely

**Architectural Alternatives**

- (A2, D0, H1)
  - Heterogeneous Multi-databases. Similar to (A1, D0, H1), but with full autonomy
- (A2, D1, H1), (A2, D2, H1)
  - Distributed Multi-database Systems

**Major DBMS Architectures**

- (Ax, D1, Hy)
  - Client Server Architecture
- (A0, D2, H0)
  - Peer-to-peer Architecture
- (A2, Dx, Hy)
  - Multi-database Architecture

**Client Server Architectures**

- Distribute the functionality between client and server to better manage the complexity of the DBMS
- Two-level Architecture

Typical Scenario
1. Client parses a query, decomposes into independent site queries, and sends to appropriate server
2. Each server processes local query and sends the result relation to client
3. Client combines the results of sub-queries

**Peer-to-peer Architecture**

- Extension of the ANSI/SPARC Architecture to include Global Conceptual Schema (GCS), where GCS is the union of all LCSs

**Multi-database Architecture**

- GCS exists as a union of some LCSs only
- or does not exist at all

User Request
Client
Request Response
Server

GCS

E1
E2
E3

LCS

E1
E2
E3

E1
E2
E3

E1
E2
E3

© School of Information Technology and Electrical Engineering
Distributed Database Design, Query and Transactions

- Topics
  - Distributed Database Design
  - Distributed DBMS
  - Distributed Query Processing
  - Distributed Transaction Management

- Readings
  - Lecture Notes
  - Oszu & Valduriez
    Selected sections of chapters 5, 7 & 12

The lecture notes for these topics contain many more slides than would actually be used for this lecture. These slides have been included for the benefit of the diverse student group.

Distributed Database Design

- Designing the Network
- Distribution of the DBMS Software
- Distribution of Application Programs
- Distribution of Data
  - Level of Sharing
  - Access Pattern
  - Level of Knowledge

Framework of Distribution

- Data + Programs
- Partial Knowledge
- Complete Knowledge
- Static
- Dynamic
- Access Pattern
- Sharing

In all cases (except no sharing) the distributed environment has new problems, not relevant in the centralized setup.

Taxonomy of Global Information Sharing

- Peer-to-peer Architecture (A0, D2, H0)
  - Local systems are homogeneous, and the global system has control over local data and processing
- Multi-database Architecture (A2, Dx, Hx)
  - Fully autonomous, heterogeneous local systems with no global schema

Design Strategies

- Bottom Up Strategy
- Top Down Strategy

Choice of Strategy depends on the Architectural Model
Top Down Design

- Objectives
- Conceptual Design
- Logical Design
- Physical Design

Bottom Up Design

- Integration of pre-existing local schemas into a global conceptual schema
- Local systems will most likely be heterogeneous
- Problems of integration and interoperability

Distributed Design Issues

- Fragmentation
  - Reasons
  - Types
  - Degree
  - Correctness
- Allocation
  - Partitioned
  - Partially Replicated
  - Fully Replicated

Why Fragment?

- Application views are subset of relations - a fragment is a subset of a relation
- If there is no fragmentation
  - Entire table is at one site - high volume of remote data accesses
  - Entire table is replicated - problems in executing updates
- Parallel execution of query, since sub-queries operate on fragments - higher concurrency

Drawbacks of Fragmentation

- Application views do not require mutually exclusive fragments - queries on multiple fragments will suffer performance degradation (costly joins and unions)
- Semantic integrity control - checking for dependency of attributes that belong to several fragments may require access to several sites

Types of Fragmentation

- Horizontal
- Vertical
- Hybrid
Horizontal Fragmentation

<table>
<thead>
<tr>
<th>PERSON</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>PHONE</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>John Smith</td>
<td>44, Here St</td>
<td>3456</td>
<td>7890</td>
</tr>
<tr>
<td></td>
<td>Alex Brown</td>
<td>1, High St</td>
<td>3678</td>
<td>1234</td>
</tr>
<tr>
<td></td>
<td>Jane Morgan</td>
<td>87, Riverview</td>
<td>8765</td>
<td>1237</td>
</tr>
<tr>
<td></td>
<td>Peter Jennings</td>
<td>65, Flag Rd</td>
<td>9851</td>
<td>1238</td>
</tr>
</tbody>
</table>

Vertical Fragmentation

<table>
<thead>
<tr>
<th>PERSON</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>PHONE</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>John Smith</td>
<td>44, Here St</td>
<td>3456</td>
<td>7890</td>
</tr>
<tr>
<td></td>
<td>Alex Brown</td>
<td>1, High St</td>
<td>3678</td>
<td>1234</td>
</tr>
<tr>
<td></td>
<td>Jane Morgan</td>
<td>87, Riverview</td>
<td>8765</td>
<td>1237</td>
</tr>
<tr>
<td></td>
<td>Peter Jennings</td>
<td>65, Flag Rd</td>
<td>9851</td>
<td>1238</td>
</tr>
</tbody>
</table>

Hybrid Fragmentation

<table>
<thead>
<tr>
<th>PERSON</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>PHONE</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>John Smith</td>
<td>44, Here St</td>
<td>3456</td>
<td>7890</td>
</tr>
<tr>
<td></td>
<td>Alex Brown</td>
<td>1, High St</td>
<td>3678</td>
<td>1234</td>
</tr>
<tr>
<td></td>
<td>Jane Morgan</td>
<td>87, Riverview</td>
<td>8765</td>
<td>1237</td>
</tr>
<tr>
<td></td>
<td>Peter Jennings</td>
<td>65, Flag Rd</td>
<td>9851</td>
<td>1238</td>
</tr>
</tbody>
</table>

Degree of Fragmentation

- **No Fragmentation**
- **Individual Tuples**
- **Individual Columns**
- **Hybrid Fragmentation**

Correctness of Fragmentation

- **Completeness**
  - Each data item found in a relation R will be found in one or more of R's fragments R₁, R₂, ..., Rₙ.
- **Reconstruction**
  - The dependency constraints on the data can be preserved by ensuring that every relation R can be reconstructed from its fragments using the operator ∇, such that R = ∇Rᵢ, ∀Rᵢ ∈ FR.
- **Disjointness**
  - A relation R is decomposed into disjoint horizontal fragments, such that if a data item cᵢ is in R, then cᵢ is in Rᵢ where j ≠ k.
  - In vertical fragments, disjointness does not apply to primary key attributes.

Integrity Constraints Issues

Consider an example: the following global schema

\[ R₁=ABC, \quad R₂=DEFG, \quad R₃=FGHIJ \]

We propose the following design:

\[ R₁₁=ABC, \quad R₁₂=ABD, \quad R₁₃=DEFG, \quad R₂₂=DF, \quad R₃₂=FGHJ \]

Obviously:

\[ R₁₁=R₂₁=R₁₂=R₂₂=R₃₂=R₃₃ \]
Let $r_2(R_2), r_3(R_3)$ be the following

$r_2(\{D \ E \ F \ G\})$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

$r_3(\{F \ G \ H \ I \ J\})$

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>0</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Integrity Constraints Issues

$r_{21}(\{D \ E \ F\})$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

$r_{22}(\{D \ G\})$

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

$r_{31}(\{F \ G \ H\})$

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

$r_{32}(\{F \ G \ I \ J\})$

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

R1 = ABCD, R2 = DEFG, R3 = FGHIJ

global schema

We execute the same query in our DDB:

We execute $r_{21}(\{D \ E \ F\})$ on Site 1 and $r_{22}(\{D \ G\})$ on Site 2.

Find all EFH

E  F  H

2  3  0
4  3  0

$2 \times r_3(\{D \ E \ F \ G \ H \ I \ J\})$

| 1 2 3 0 2 0 | 4 3 0 |

Find all EFH

E  F  H

2  3  0
4  3  0

Often, more constraints must be maintained

Integrity Constraints Issues

Example of Derived HFs

EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEPT</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>CSEE</td>
<td>34000</td>
</tr>
<tr>
<td>Alex Brown</td>
<td>CHEMISTRY</td>
<td>48000</td>
</tr>
<tr>
<td>Harry Potter</td>
<td>CHEMISTRY</td>
<td>98000</td>
</tr>
<tr>
<td>Jane Morgan</td>
<td>CSEE</td>
<td>65800</td>
</tr>
<tr>
<td>Peter Jennings</td>
<td>CSEE</td>
<td>23980</td>
</tr>
<tr>
<td>Melissa Dean</td>
<td>ARCHIT</td>
<td>60000</td>
</tr>
</tbody>
</table>

DEPT

<table>
<thead>
<tr>
<th>DEPT</th>
<th>BUILDING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSEE</td>
<td>GP-SOUTH</td>
</tr>
<tr>
<td>CHEMISTRY</td>
<td>HAWKEN</td>
</tr>
<tr>
<td>ARCHIT</td>
<td>HAWKEN</td>
</tr>
</tbody>
</table>

EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEPT</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>CSEE</td>
<td>34000</td>
</tr>
<tr>
<td>Alex Brown</td>
<td>CHEMISTRY</td>
<td>48000</td>
</tr>
<tr>
<td>Harry Potter</td>
<td>CHEMISTRY</td>
<td>98000</td>
</tr>
<tr>
<td>Jane Morgan</td>
<td>CSEE</td>
<td>65800</td>
</tr>
<tr>
<td>Peter Jennings</td>
<td>CSEE</td>
<td>23980</td>
</tr>
<tr>
<td>Melissa Dean</td>
<td>ARCHIT</td>
<td>60000</td>
</tr>
</tbody>
</table>

Relations

Fragments: Co-located employees

Versions of HFs

• Primary Horizontal Fragmentation
  – Selection operation on the owner relation of a database schema
  – SELECT * FROM owner WHERE m

• Derived Horizontal Fragmentation
  – Defined on the member relation of a link according to the selection operation specified for its owner
  – SELECT * FROM member WHERE member.join-attr IN (SELECT owner.join-attr FROM owner WHERE m)
**Approaches for VFs**

- Vertical fragmentation is more complex than horizontal fragmentation due to the total number of alternatives available
  - HF: number of minterm predicates
  - VF: number of non-primary key attributes
- Heuristic Approaches
  - Grouping: Bottom Up
  - Splitting: Top Down

**Allocation Problem**

Find the optimal distribution of \( F = \{F_1, F_2, \ldots, F_n\} \) to \( S = \{S_1, S_2, \ldots, S_m\} \), on which a set of applications \( Q = \{q_1, q_2, \ldots, q_q\} \) are running.

Optimality can be defined through:

<table>
<thead>
<tr>
<th>Minimal Cost</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost of storing and querying a fragment at a site</td>
<td>Cost of updating a fragment at all sites</td>
</tr>
<tr>
<td>Cost of communicating data</td>
<td>Maximize response time</td>
</tr>
</tbody>
</table>

**Guidelines for Distributed Design**

- Given a database schema, how to decompose it into fragments and allocate to distributed sites, while minimizing remote data access and update problems
  - Most active 20% of queries account for 80% of the total data accesses
  - Qualitative aspect of the data guides the fragmentation activity
  - Quantitative aspect of the data guides the allocation activity

**Information Requirements**

- Fragmentation
  - Database Information
  - Application Information
- Allocation
  - Communication Network Information
  - Computer System Information

**Information requirements for HF**

- Database Information
  - Links between relations (relations in RDB)
- Application Information
  - Predicates in user queries
    - Simple: \( \pi \text{ DEPT}=\text{CSEE}, \pi \text{ SAL}>30000 \)
    - Conjunctive: \( minterm \text{ predicate: } m_{p_1,p_2,p_3,DEPT=\text{CSEE} \land \text{SAL}>30000} \)
  - Minterm selectivity: Number of tuples returned against a given minterm
  - Access Frequency: Access frequency of user queries and/or minterms
Information requirements for VF

- **Application Information**
  - Place in one fragment, attributes *usually* accessed together.
  - **Affinity**: The bond between two attributes on the basis of how they are accessed by applications.
  - Affinity is measured by Usage/Affinity Matrix
    - Usage of an attribute A in query q: 
      \[ \text{Use}(q, A) = \begin{cases} 1 & \text{if used} \\ 0 & \text{if not used} \end{cases} \]
    - Affinity of two attributes Ai and Aj for a set of queries:
      \[ \text{Aff}(Ai, Aj) \]

Information Requirements for Allocation

- **Database Information**
  - Fragment Selectivity: The number of tuples of a fragment Fi that need to be accessed in order to process a query qi.
  - The size of a fragment: 
    \[ \text{Size}(Fi) = \text{Cardinality}(Fi) \times \text{Length-in-bytes}(Fi) \]

- **Network Information**
  - Channel capacities, distances, protocol overhead
  - Communication cost per frame between sites Si and Sj

Distributed DBMS

- Components of Distributed DBMS
- Query Processing and Optimization
- Transaction Management
- Concurrency Control
- Reliability Issues

Distributed Query Processing

- The Distributed Query Problem
- Layers of Query Processing
- Query Decomposition
- Data Localization
- Query Optimization
Distributed Query Processing

Distributed Query Processor
– Map high level query on a distributed database, into a sequence of database operations on fragments
– Database Operations + Communication Operations

Distributed Query Optimizer
– Minimization of cost function based
  • Computing resources (CPU, I/O)
  • Communication networks

A Simple Query Problem

EMP (ENO, ENAME, TITLE)
ASG (ENO, PNO, RESP, DUR)

Find the names of employees who are managing a project

SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
AND RESP = 'Manager';

A Distributed Query Problem

EMP and ASG are fragmented as
EMP1 = σ ENO ≥ 'E3' (EMP)
EMP2 = σ ENO < 'E3' (EMP)
ASG1 = σ ENO ≥ 'E3' (ASG)
ASG2 = σ ENO < 'E3' (ASG)

ASG1, ASG2, EMP1, EMP2 are stored at sites 1, 2, 3, and 4. The result is expected at site 5!

Equivalent Strategies

Strategy A
Total Cost = 460

Strategy B
Total Cost = 23000

Total cost is determined through tuple accesses and transfers

Layers of Query Processing

Control Site
– Control site receives the query
– Query decomposition

Local Sites
– Local sites perform local queries
– Optimized local queries

Go to 97
**Query Decomposition**
The same techniques as in centralized since information on data distribution is not used

- **Step 1:** Normalization
- **Step 2:** Analysis
- **Step 3:** Elimination of Redundancy
- **Step 4:** Rewriting

**Step 1: Normalization**
- Transform arbitrary complex queries into a normal form
- Two Normal Forms
  - Conjunctive: \((P_1 \land P_2 \land \ldots) \land (P_m)\)
  - Disjunctive: \((P_1) \lor (P_2) \lor (P_m)\)
- Example equivalence rules
  - \(P_1 \land (P_2 \lor P_3) \equiv (P_1 \land P_2) \lor (P_1 \land P_3)\)
  - \(\neg(P_1 \land P_2) \equiv \neg P_1 \land \neg P_2\)
- Example transformation
  - \(\text{TITLE} = 'Manager' \land \text{SAL} > 30000 \land \text{PROJ} = 12 \lor \text{PROJ} = 34\)

**Step 2: Analysis**
- Determine if further processing of the normalized query is possible.
  - Type incorrectness
    - If attributes or relation names are not defined in the global schema
    - If operations are being applied to attributes of the wrong type
  - Semantic incorrectness
    - If components do not contribute to the generation of the result
    - Query graphs - disconnected graphs show incorrectness

**Step 3: Elimination of Redundancy**
- Simplification of the query qualification (WHERE clause) using well known idempotency rules
  - \(P \land \text{false} \equiv \text{false}\)
  - \(P \lor \text{P} \equiv P\)
  - \(P \lor (P_1 \vee P_2) \equiv P_1\)
  - \(P_1 \land \neg P \equiv \text{false}\)
  - etc.
- Example redundancy
  - \(\text{SAL} > 30000 \land \neg(\text{SAL} \leq 3000) \equiv \text{SAL} > 30000\)

**Step 4: Rewriting**
- **Step 4.1:** Transformation
  - Transform relational calculus query into relational algebra using Operator Tree
  - By applying Transformation Rules many different but equivalent trees may be found
- **Step 4.2:** Restructuring
  - Restructuring the tree to improve performance

**Data Localization**
- Localize the query using data distribution information
- Algebraic query on global relations
  - \(\rightarrow\) Algebraic query on physical fragments
- Two Step Process
  - Localization
  - Reduction
### Data Localization

- **Step 1:** Map distributed query to fragment query using *Localization* program
  
  \[ \text{EMP1} = \sigma_{\text{SAL} > 30000} (\text{EMP}) \text{ and EMP2} = \sigma_{\text{SAL} \leq 30000} (\text{EMP}) \]
  
  then localization program for \( \text{EMP} = \text{EMP1} \cup \text{EMP2} \)

- **Step 2:** Simplify and restructure the fragment query using *Reduction* techniques
  
  Basic Idea is to determine the intermediate results in the query that produce empty (horizontal) or useless (vertical) relations, and remove the subtrees that produce them

### Query Optimization

- **Query optimization has been independent of fragment characteristics such as cardinality and replication**
  
  - Main objectives of the optimizer (NP-hard !)
    - Find a strategy close to optimal
    - Avoid Bad strategies
  
  - Choice of optimal strategy requires prediction of execution costs
    - Local Processing (CPU, I/O) + Communication

### Distributed Query Optimization

- **Global Optimization**
  
  - Join Ordering
  
  - Semijoin based Algorithms

- **Local Optimization**
  
  - INGRES Algorithm
  
  - System R Algorithm

  Most try to optimize the ordering of joins

### Components of the Optimizer

- **Search Space**
  
  - The set of alternative, but equivalent query execution plans
  
  - Abstracted by means of operator trees

- **Search Strategy**
  
  - Explores the search space and selects the best plan using the cost model
  
  - Defines how the plans are examined

- **Distributed Cost Model**
  
  - Predicts the cost of a given execution plan

### Distributed Cost Model

- **Cost Functions**
  
  - Total Time = \( T_{\text{CPU}} \times \#\text{insts} + T_{\text{I/O}} \times \#\text{I/Os} + T_{\text{MSG}} \times \#\text{msgs} + T_{\text{TR}} \times \#\text{bytes} \)

  - Response Time = \( T_{\text{CPU}} \times \text{seq-\#insts} + T_{\text{I/O}} \times \text{seq-\#I/Os} + T_{\text{MSG}} \times \text{seq-\#msgs} + T_{\text{TR}} \times \text{seq-\#bytes} \)

- **Database Statistics**
  
  - Estimate the size of the intermediate relations
  
  - Based on statistical information: Length of attributes, distinct values, number of tuples in the fragment ...

### Cardinalities of intermediate relations

- **Formulas for estimating the cardinalities of the results of the basic relational algebra operations**
  
  - Selection, Projection, Cartesian product, Join, Semi Join, Union, Difference

- **Examples**
  
  - \( \text{card} (\Pi (R)) = \text{card} (R) \)
  
  - \( \text{card} (R \times S) = \text{card} (R) \times \text{card} (S) \)
Semijoins

- The dominant factor in the time required to process a query is the time spent on data transfer between sites (not the computation or retrieval from secondary storage).
- Example: suppose that table r is stored at site 1, and s at 2 to evaluate (r join σA=a(s)) at site 1.

1) site 1 asks site 2 to transmit s and then computes the selection itself.
2) site 1 transmits the selection condition to site 2 and site 2 sends selected records to site 1 (presumably much smaller than s).

Semijoin Example

1) We can transfer s to site 1 and compute join at site 1 (24 values).
2) We compute r' = \( \Pi_B(r) \) at site 1, send r' to site 2, compute s' = (r' join s) and send s' to site 1 (15 values).
3) We compute (r \( \bowtie \) s) as (r \( \bowtie \) s')

Thus, r \( \bowtie \) s can be computed knowing only s'.

Semijoin

Definition: let r(R) and s(S) be two relations. The semijoin of r with s denoted \( r \bowtie s \) is the relation \( \Pi_r(\text{proj}_r(s)) \), that is, r \( \bowtie s \) is the portion of r that joins with s.

Some important transformations:

\[ r \bowtie (s \bowtie t) = \Pi_r((\text{proj}_r(s)) \bowtie (\text{proj}_r(t))) \]

Distributed Transaction Management

- Transaction Management - Revisited
- Distributed Transactions
- Distributed Execution Monitor
- Advanced Transaction Models

Centralized transaction processing

Data Access:
read-item (X), write-item (X)

- Buffers

Interleaving

Transaction States

Active

Partially Committed

Committed

Dead

Aborted

Terminated

Well-defined Repetitive

High volume

Update

Controlled

Lecture Notes INFS7907
© School of Information Technology and Electrical Engineering
Desirable properties
ACID properties - enforced by concurrency control and recovery methods of the DBMS
– Atomicity
– Consistency
– Isolation
– Durability

Distributed Transactions
• Database Consistency
  – No semantic integrity constraints are violated
• Transaction Consistency
  – No incorrect updates during concurrent database accesses
• Mutual Consistency
  – No difference in the values of a data item in all the replicas to which it belongs

Example
Consider the following example:
• A chain of department stores has many individual stores.
• Each store has a database of sales and inventory at that store.
• There may also be a central office database with data about employees, chain-wide inventory, credit card customers, information about suppliers such as outstanding orders etc.
• There is a copy of all the stores’ sales data in a data warehouse, which is used to analyse and predict sales

Consider the following example:
• A chain of department stores has many individual stores.
• Each store has a database of sales and inventory at that store.
• There may also be a central office database with data about employees, chain-wide inventory, credit card customers, information about suppliers such as outstanding orders etc.
• There is a copy of all the stores’ sales data in a data warehouse, which is used to analyse and predict sales

Example
A manager wants to query all the stores, find the inventory of toothbrushes and have them moved from store to store in order to balance the inventory
This transaction will have a local component T at the i-th store, and a global component T0 at the central office:
1. Component T0 is created at the central office
2. T0 sends message to all stores instructing them to create Ti
3. Each Ti executes query to determine number of toothbrushes and returns result to T0
4. T0 takes these numbers, determines (by some algorithm) the shipments required, and sends instructions to all affected stores. For example “Store 10 to ship 500 toothbrushes to store 7”
5. Stores receiving these instructions update their inventory and perform the shipment

What can go wrong?
A bug in the algorithm causes store 10 to ship more toothbrushes than available in its inventory causing T10 to abort. However, T7 detects no problem and hence updates its inventory in anticipation of the incoming shipment and commits
• Atomicity of T is compromised (T10 never completed)
• Database is left in an inconsistent state
After returning the result (step 3) of its inventory count to T0, the machine at store 10 crashes. The instructions to ship are not received from T0.
• What should T10 do when its site recovers?

Problems
• Managing the commit/abort decision in distributed transactions?
  (One component wants to abort while others want to commit)
  • Two phase commit
• Assuring serializability of transactions that involve components at several sites?
  • Distributed locking
Commit in Centralized Databases

All database access operations of the transaction have been executed successfully and their effect on the database has been recorded in the log.

System log entries:
[start-transaction, T];
[read-item, T, X];
[write-item, T, X, old-value, new-value];
[commit, T];

After the commit point, the transaction T is said to be committed.

Commit in Distributed Databases?

Distributed DBMSs may encounter problems not found in a centralized environment:
- Dealing with multiple copies of the data
- Failure of an individual site
- Failure of communication links

Recording the effect of database access operations in the (local) system log is not a sufficient condition for distributed commit.

A Distributed Execution Monitor

- **Begin-transaction**
  - Basic housekeeping by TM
- **Read**
  - If data item is local, then TM retrieves, else selects one copy
- **Write**
  - TM coordinates the updating of the data item value at all sites where it resides
- **Commit**
  - TM coordinates the physical updating of all databases that contain a copy of each data item for which the write was issued
- **Abort**
  - TM ensures that no effect of the transaction is reflected in the database

Classification of Transactions

- **Distribution**
  - Transactions in Centralized DBMSs
  - Transactions in Distributed DBMSs
- **Duration**
  - Short-life transactions
  - Long-life / long duration transactions
- **Processing**
  - On-line / interactive transactions
  - Batch transactions
- **Grouping of Operations**
  - Flat transactions
  - Nested transactions

Selected Adv. Transaction Models

- **Nested Transactions**:
  - Grouping of operations into hierarchical structures
- **Workflow (long duration) Transactions**:
  - Relaxation of the ACID Properties

Nested Transactions

A set of sub-transactions that may recursively contain other sub-transactions

- Begin-transaction Reservation
  - Begin-transaction Airline
    - End [Airline]
  - Begin-transaction Hotel
    - End [Hotel]
  - Begin-transaction Car
    - End [Car]
  - End
Types of Nested Transactions

• Closed Nested Transaction
  – Sub-transaction begins after the root and finishes before
  – Commit of sub-transaction is conditional upon the commit of the root
  – Top-level atomicity

• Open Nested Transactions
  – Relaxation of top-level atomicity
  – Partial results of sub-transactions visible
    • Sagas
    • Split transactions

Advantages of Nested TM

• Higher level of concurrency
  – Objects can be released after sub-transaction

• Independent recovery of sub-transactions
  – Damage is limited to a smaller part, making it less costly to recover

• Creating new transactions from existing ones

Workflow (long duration) Transcts

• Generic definition:
  – A workflow is an automation of a business process

• In the context of transactions:
  – Workflow is a (high level) activity, that consists of a set of tasks with a well-defined precedence relationship between them

• Workflow tasks (sub-transactions) are allowed to commit individually (open nesting semantics), permitting partial results to be visible outside the workflow

• Various concepts introduced to overcome problem of dealing with sub-transaction commit
  – Compensating tasks
  – Critical tasks
  – Contingency tasks

Sagas: Open and Long-duration Transactional Model

• A collection of actions that form a long duration transaction
  – A collection of actions
  – A graph whose nodes are either actions, or one of {Abort, Complete} called the terminal nodes
  – An indication of the start called the start node

Successful paths:
A0, A1, A2, A3

Unsuccessful paths:
A0, A1, A4
A0, A1, A2, A5

Concurrency Control in Sagas

Concurrency control is managed by two facilities

1. Each action “A”, is itself a (short) transaction
   • Uses conventional concurrency control such as locking

2. The overall transaction which can be any path from the start node to one of the terminal nodes
   • Uses the concept of Compensating transactions
   • A Compensating transaction “rolls back the effect of a committed action in a way that does not depend on what happened to the database between the time of the action and the time of the compensating transaction
   • If A is any action, A\(^{-1}\) is its compensating transaction, and it is any sequence of legal actions and compensating transactions, then A \(\alpha\) \(A^{-1}\) = \(\alpha\)

Distributed Concurrency Control

• Distributed Concurrency Control through Timestamps
• Distributed Concurrency Control through Locking
Centralized Timestamp Ordering

- For each item accessed by conflicting operations in a schedule, the order of item access does not violate the serializability order.
- Each item has:
  - read-TS (X) = TS (T)
  - write-TS (X) = TS (T)
  where T is the transaction that most recently read (wrote) X

Distributed Timestamp Ordering

TM is responsible for assigning a timestamp to each new transaction and attaching it to each database operation passed to the scheduler. SC is responsible for keeping track of read write timestamps and performing serializability checks.

Centralized 2 Phase Locking

A transaction follows the two-phase protocol if all locking operations precede the first unlocking operation.

Phase 1: Growing
- read-lock (X)
- write-lock (X)
- write-lock (Y)

Phase 2: Shrinking
- unlock (X)
- unlock (Y)

Distributed 2 Phase Locking

- TM sends request to lock managers of ALL participating sites
- Operations are passed to data processors by local as well as REMOTE participating lock managers

Distributed Locking

Main difference from Centralized Locking
- Replication Control
- Logical Elements --- Global Locks

How do we manage locks for objects across many sites?
- Centralized
- Primary Copy
- Fully Distributed

Centralized 2PL

- One site is designated as the lock site or primary site
- The primary site is delegated the responsibility for lock management
- Transaction managers at participating sites deal with the primary site lock manager rather than their own

Differences
- TM sends request to lock managers of ALL participating sites
- Operations are passed to data processors by local as well as REMOTE participating lock managers

Lecture Notes INFS7907
© School of Information Technology and Electrical Engineering
**Primary Copy 2PL**

- The function of the primary site is distributed (thus avoiding bottleneck at the primary site)
- Lock managers at a number of sites
- Each lock manager responsible for managing locks for a given set of data elements
- The copy of the data element at this site is called the primary copy, that is each logical element still has a single site responsible for maintaining global lock
- Access to the data element will require access to transaction site as well as the primary copy site

**Distributed 2PL**

- Lock managers at each site
- If no replication, Distributed 2PL is the same as Primary copy 2PL
- When there is replication, a transaction must acquire global lock (No copy of a data element is primary)

When can a transaction assume it has a (shared or exclusive) global lock on a data element?

**Distributed CC Based on Voting**

- Majority Locking
  - If a data item Q is replicated in n different sites, then a lock-request message must be sent to more than one-half of the n sites in which Q is stored.
  - The requesting transaction does not operate on Q until it has obtained a lock on a majority of replicas for Q
- Read One, Write All Locking
  - When a transaction needs to read/shared lock a data item Q, it requests a lock from the lock manager at one site that contains a replica of Q
  - When a transaction needs to write/exclusive lock a data item Q, it requests a lock on Q from the lock manager of all sites that contain a replica of Q

**Deadlocks - Revisited**

Locking-based concurrency control may result in deadlocks!

Deadlock: Each transaction in a set (of >2 transactions) is waiting for an item which has locked another transaction in the set

**Distributed Deadlock Detection**

- Each site maintains a local waits-for graph.
- A global deadlock might exist even if the local graphs contain no cycles:

  SITE A
  
  SITE B

**Distributed Deadlock Detection**

- Three Solutions:
  - Centralized
    - One site designated as deadlock detector
    - Lock managers at other sites periodically send local wait-for graphs to deadlock detector, that forms a global wait-for graph
    - Length of interval is a system design decision
    - Approach vulnerable to failure and has high communication overhead
  - Hierarchical
  - Timeout
**Distributed Deadlock Detection**

Three Solutions:
- Centralized
- Hierarchical
  - Sites are organized into a hierarchy of deadlock detectors
  - Local graphs are sent to parent in the hierarchy
  - Reduces communication overhead and dependence on central deadlock detection site
- Timeout

**Distributed Reliability Issues**

- Failures in Distributed Systems
- Distributed Reliability Protocols

**Centralized Recovery**

- Main Techniques
  - Deferred Update
  - Immediate Update
- Concepts
  - Disk Caching
  - System Log
  - Checkpointing

**Failures in Distributed Systems**

- Transaction Failures
- Site (System) Failures
- Media Failures
- Communication Failures

**Local Recovery Manager**
Distributed Reliability Protocols

- **Commit Protocols**
  - Maintain atomicity of distributed transactions.
  - **Atomic Commitment**: Ensures that the effects of the transaction on the distributed database is all-or-nothing, even though the distributed transaction may involve multiple sites.

- **Recovery Protocols**
  - The procedure a failed site has to go through to recover its state after restart.
  - **Independent Recovery Protocols**: Protocols that determine how to terminate a transaction that was executing at a failed site without having to consult any other site.

- **Termination Protocols**
  - The procedure that the operational sites go through when a participating site fails.
  - **Non-blocking Termination Protocols**: Protocols that determine how to allow the transaction at the operational site to terminate, without waiting for the failed site to recover.

---

Commit Protocols

1 Phase Commit (1PC)
- **Ready**: All operations of the transaction completed except commit.

2 Phase Commit (2PC)
- **Prepare**: Global Commit Rule
  - Global Abort: If one participant votes to abort.
  - Global Commit: If all participants vote to commit.

---

Centralized 2P Commit Protocols

**Phase 1**
- **Prepare Vote**
- **Coordinator**
- **Participants**

**Phase 2**
- **Global Commit**
  - Decision made independently.

---

Distributed 2P Commit Protocols

**Phase 1**
- **Prepare**
  - **Coordinator**
  - **Participants**
- **Vote**
- **Global Commit**

**Phase 2**
- **Commit**

**Comments on 2PC**

- Two rounds of communication: first, voting; then, termination. Both initiated by coordinator.
- Any site can decide to abort an Xact.
- Every msg reflects a decision by the sender; to ensure that this decision survives failures, it is first recorded in the local log.
- All commit protocol log recs for an Xact contain Xactid and Coordinatorid. The coordinator’s abort/commit record also includes ids of all subordinates.
**Restart After a Failure at a Site**

- If we have a commit or abort log rec for Xact T, but not an end rec, must redo/undo T.
  - If this site is the coordinator for T, keep sending commit/abort msgs to subs until acks received.
- If we have a prepare log rec for Xact T, but not commit/abort, this site is a subordinate for T.
  - Repeatedly contact the coordinator to find status of T, then write commit/abort log rec; redo/undo T; and write end log rec.
- If we don’t have even a prepare log rec for T, unilaterally abort and undo T.
  - This site may be coordinator! If so, subs may send msgs.

**Blocking**

- If coordinator for Xact T fails, subordinates who have voted yes cannot decide whether to commit or abort T until coordinator recovers.
  - T is **blocked**.
  - Even if all subordinates know each other (extra overhead in prepare msg) they are blocked unless one of them voted no.

**Link and Remote Site Failures**

- If a remote site does not respond during the commit protocol for Xact T, either because the site failed or the link failed:
  - If the current site is the coordinator for T, should abort T.
  - If the current site is a subordinate, and has not yet voted yes, it should abort T.
  - If the current site is a subordinate and has voted yes, it is blocked until the coordinator responds.

**2PC with Presumed Abort**

1. Ack msgs used to let coordinator know when it can “forget” an Xact (T is kept in the Xact Table until it receives all acks)
2. If coordinator fails after sending prepare msgs but before writing commit/abort log recs, when it comes back up it aborts the Xact.
   - In the absence of information, T is presumed to have aborted
     - When coordinator aborts T, it undos T and removes it from the XactTable immediately, resulting in a no information state
     - Similarly, subordinates do not need to sandacks on abort. If it had voted yes previously, then a later inquiry will result in no information and consequently abort.
     - Names of subs not recorded in abort log rec.
3. If a subtransaction does no updates, its commit or abort status is irrelevant.
   - If subXact does not do updates, it responds to prepare msg with reader instead of yes/no.
   - Coordinator subsequently ignores readers.
   - If all subXacts are readers, 2nd phase not needed.

**Commit Protocols**

- 3 Phase Commit Protocol – avoids the possibility of blocking in a restricted case of possible failures:
  - no network partition can occur,
  - at any point at least one site must be up,
  - at any point at most K participating sites can fail,

  Interesting but optional for this course

**Distributed Recovery**

- Two new issues:
  - New kinds of failure, e.g., links and remote sites.
  - If “sub-transactions” of an Xact execute at different sites, all or none must commit. Need a commit protocol to achieve this.
- A log is maintained at each site, as in a centralized DBMS, and commit protocol actions are additionally logged.
Next Lecture

Data warehousing and OLAP
Technologies