Lecture 5: Distributed Database

Haibo Zhang

Computer Science, University of Otago
Learning Objectives

• You should be able to
  – define the key concepts in distributed database
  – distinguish between different types of distributed database
  – understand the architectures of distributed database
  – explain how to perform data fragmentation, allocation, and replication
  – explain how to do semijoin in distributed database
  – understand technologies for distributed transactions

• You will get a taste of carrying out scientific research
  – optimization on data allocation and replication
  – the study of the research paper on Google Spanner
Definitions

• Distributed database
  – a collection of multiple, *logically interrelated* databases distributed over a *computer network*

• Distributed DBMS
  – a software system that manages a distributed database and provides an access mechanism while making the distribution *transparent to the users*.

Distributed database system = Database + Communication
Key Concepts

• Data stored at several locations
  – Fragmentation
  – Replication

• **Transparency**: hide implementation for end users/application developers
  – Location
  – Fragmentation
  – Replication
  – Design
  – Execution

```sql
SELECT *
FROM employee
WHERE dno = 4;
```
Types of Distributed Databases

- Homogeneous
  - All sites run the same DBMS

- Heterogeneous
  - Different sites can run different DBMSs.
Distributed Database Architecture

- Client-Server
  - a two-level architecture based on the client-server concept (database centric)
  - A client can directly or indirectly connect to a server

```
TRANSACTION
INSERT INTO EMP@SALES...;
DELETE FROM DEPT...;
SELECT... FROM EMP@SALES...;
COMMIT;
```
Distributed Database Architecture

• Peer-to-Peer
  – All nodes have the same role and functionality
  – High scalability and flexibility
  – Hard to manage because all machines are autonomous and loosely coupled

![Distributed Database Architecture Diagram]
Main Issues

- Data layout
  - Data fragmentation
  - Data allocation and replication
- Query processing and optimization
  - Data transfer cost
  - Semijoin
  - Query and update decomposition
- Distributed transactions
  - Transaction atomicity using two-phase commit
  - Transaction serializability using distributed locking
Data Fragmentation

• Break up the database into logical units, called **fragments**, which may be assigned for storage at various sites

• What is a reasonable unit of data distribution?
  – Relation
    • Views are subset of relations
    • Extra communications
    • Less parallelism
  – Sub-relations
    • Extra processing for views that cannot be defined on a single fragment
    • Better parallelism
    • Difficult to enforce integrity constraints
Data Fragmentation

• Horizontal fragmentation
  – Grouping rows to create subsets of tuples
  – Each subset has a certain logical meaning

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorganisation</td>
<td>20</td>
<td>Houston</td>
<td>1</td>
</tr>
<tr>
<td>Computerisation</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>NonProject</td>
<td>40</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>ProductX</td>
<td>1</td>
<td>Bellaire</td>
<td>5</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>Houston</td>
<td>5</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>Sugarland</td>
<td>5</td>
</tr>
</tbody>
</table>

PROJ₁

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computerisation</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>NonProject</td>
<td>40</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

PROJ₂

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>Bellaire</td>
<td>5</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>Sugarland</td>
<td>5</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>Houston</td>
<td>5</td>
</tr>
</tbody>
</table>

PROJ₃
Data Fragmentation

- Vertical fragmentation
  - Divide the relation by columns
  - Each fragment has the primary key or some candidate key so that the full relation can be reconstructed
Data Fragmentation

• Mixed (Hybrid) fragmentation
  – Intermix the two types of fragmentation

• Correctness of fragmentation
  – **Completeness:** each data item can be found in one fragment
  – **Reconstruction (lossless):** the full relation can be reconstructed from all fragments
  – **Disjointness (Non-overlapping):** each data item except the key should not be in more than one fragment
Data Replication

• Non-replicated
  – Each fragment resides at only one site

• Replicated
  – Fully replicated: each fragment at each site
  – Partially replicated: each fragments at some sites

• Pros & Cons
  – Improve availability, distribute load, cheaper reads
  – Complexity on update and storage

• Rule of thumb
  If \( \frac{\text{read-only queries}}{\text{update queries}} \geq 1 \), replication is advantageous, otherwise replication may cause problem
Data Replication

• Non-replicated
  – Each fragment resides at only one site
• Replicated
  – Fully replicated: each fragment at each site
  – Partially replicated: each fragments at some sites
• Pros & Cons
  – Improve availability, distribute load, cheaper reads
  – Complexity on update and storage
• Rule of thumb

\[
\frac{\text{read-only queries}}{\text{update queries}} \geq 1, \text{ replication is advantageous, otherwise replication may cause problem}
\]
## Comparison of Replication Alternatives

<table>
<thead>
<tr>
<th></th>
<th>Full-replication</th>
<th>Partial replication</th>
<th>Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query processing</td>
<td>Easy</td>
<td>Same difficulty</td>
<td></td>
</tr>
<tr>
<td>Directory management</td>
<td>Easy or Non-existant</td>
<td>Same difficulty</td>
<td></td>
</tr>
<tr>
<td>Concurrency control</td>
<td>Moderate</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
<tr>
<td>Reliability</td>
<td>Very high</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
Optimization

• What is the best fragmentation, replication and allocation?
• A taste on allocation optimization

A genetic algorithm for fragment allocation in a distributed database system (Corcoran, SAC’94)

A distributed database with a collection $S$ of $m$ sites, where each site $i$ has a capacity $c_i$

$$S = \{c_1, c_2, c_3, ..., c_i, ..., c_m\}$$

A set $F$ of $n$ fragments where each fragment $j$ has a size of $s_j$

$$F = \{s_1, s_2, s_3, ..., s_j, ..., s_n\}.$$ 

Site requirement matrix, where $r_{ij}$ indicates the requirement by site $i$ for fragment $j$

$$R = \begin{bmatrix}
    r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\
    r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\
    \vdots & \vdots & \ddots & \vdots \\
    r_{m,1} & r_{m,2} & \cdots & r_{m,n}
\end{bmatrix}$$
A Taste on Allocation Optimization

Transmission cost matrix, where \( t_{i,j} \) indicates the cost by site \( i \) to access a fragment on site \( j \)

\[
T = \begin{bmatrix}
    t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\
    t_{2,1} & t_{2,2} & \cdots & t_{2,m} \\
    \vdots & \vdots & \ddots & \vdots \\
    t_{m,1} & t_{m,2} & \cdots & t_{m,m}
\end{bmatrix}
\]

Fragment placement vector \( P = \{p_1, p_2, p_3, \ldots, p_j, \ldots, p_n\} \)

where \( p_j = i \) indicates fragment \( j \) is located at site \( i \).

The objective is to minimize the total transmission cost.

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{m} \sum_{j=1}^{n} r_{i,j} t_{i,p_j} \\
\text{Subject to} & \quad \sum_{j=1}^{n} r_{i,j} s_j \leq c_i, \quad \forall i | 1 \leq i \leq m
\end{align*}
\]
Distributed Query Processing

- **Optimization goal:** reduce the amount of data transfer

  - **Option 1:** Send both $R$ and $S$ to Site 1 for join
  - **Option 2:** Send $R$ to Site 3 to join, send join results back to site 1
  - **Option 3:** Send $S$ to Site 2 to join, send join results back to site 1

Which option is better? Is there another better option?
Semi-Join

- **Idea:** reduce the number of tuples in a relation before transferring it to another site.

  - Site 3 sends only S.Y column to site 2
  - Site 2 does the join based on Y column in R’s location, and send the records of R that will join (without duplicates) back to Site 3
  - Site 3 perform the final join
Semi-Join Example

F = \prod_{Dnumber} (Employee)

Q = \prod_{Dname, Dnumber} (F \bowtie_{Dno=Dnumber} Department)

\Pi_{Fname, Lname, Dname} (Employee \bowtie_{Dno=Dnumber} Department)
Query or Update Decomposition

- **Idea:** decompose a query or update into a sequence of queries or updates that can be executed at the individual sites

\[ \Pi_{\text{Fname, Lname, hours}} \sigma_{\text{dno}=5} (\text{Employee} \bowtie_{\text{Dno} = \text{Dnumber}} \text{Project} \bowtie_{\text{Dno} = \text{Dnumber}} \text{Works_on}) \]

Suppose a site stores all information about the projects controlled by department 5 (Projs5) and the employees who work on these projects (Works_on5).

\[ T_1 \leftarrow \Pi_{\text{Essn, pno, hour}} \sigma_{\text{dno}=5} (\text{Projs5} \bowtie_{\text{Pno} = \text{Pnumber}} \text{Works_on5}) \]

\[ T_2 \leftarrow \Pi_{\text{Essn, Fname, hours}} (T_1 \bowtie_{\text{Ssn} = \text{Essn}} \text{Employee}) \]
Transactions

• A Transaction is an atomic sequence of actions (reads and writes)

• ACID properties
  – Atomicity
  – Consistency
  – Isolation
  – Durability
Transactions Management

• One transaction T may touch many sites
  – T={T1, T2, ..., Tn}
  – Tk runs at site k
  – How to guarantee atomicity?

• Two-Phase Commit protocol
  – Global transaction manager or coordinator
  – The site initiated the transaction can be the coordinator
Two-Phase Commit (2PC) Protocol

- **Request phase**
  - Coordinator sends a *prepare message* to all sites
  - Each site replies with “Ready/Abort” in the *vote message*

- **Commit phase**
  - Coordinator sends a commit message if each site vote “Ready”, otherwise sends an abort message
  - Each site either commits or aborts, and replies with an acknowledge message
Three-Phase Commit (3PC) Protocol

- 2PC cannot recover from a failure of both the coordinator and a site during the commit phase.
- Solved in 3PC by introducing the Prepared to commit state.

---

Coordinator

<table>
<thead>
<tr>
<th>Soliciting votes...</th>
<th>canCommit?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Site

<table>
<thead>
<tr>
<th>Commit authorized.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Finalizing commit.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Done.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Phase 1

- Uncertain.
- Timeout causes abort.

Phase 2

- Prepared to commit.
- Timeout causes commit.

Phase 3

- Committed.
Concurrency Control in Distributed DB

• More complicated than that in a centralized DB environment
  – Dealing with multiple copies of the data item
  – Failure of individual site
  – Failure of communication links
  – Distributed commit
  – Distributed deadlock
Distributed Locking

• Centralized approach
  – One dedicated site manages all locks
  – Bottleneck, unsalable, single point failure

• Primary-copy approach
  – Each data item has a primary site.
  – Each transaction asks the primary site for lock on the data item.

• Full distributed (voting) approach
  – A lock request is sent to all sites with a copy of the data item
  – Each copy maintains its own lock and can grant or deny the request for it
  – Use timeout to resolve deadlock
Summary

• Data fragmentation, allocation, and replication is a key issue that affect the performance of a distributed database

• Query in distributed database can be optimized based on semi-join

• Transaction management and concurrency control are more complex than those in centralized database