Chapter 19: Distributed Databases

Database System Concepts, 6th Ed.

Chapter 19: Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Distributed Query Processing
- Heterogeneous Distributed Databases

Distributed Database System

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites

Homogeneous Distributed Databases

- In a homogeneous distributed database
  - All sites have identical software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Appears to user as a single system
- In a heterogeneous distributed database
  - Different sites may use different schemas and software
    - Difference in schema is a major problem for query processing
    - Difference in software is a major problem for transaction processing
  - Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Chapter 19: Distributed Databases

- Heterogeneous and Homogeneous Databases
- **Distributed Data Storage**
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Distributed Query Processing
- Heterogeneous Distributed Databases

Distributed Data Storage

- Assume relational data model
- Replication
  - System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.
- Fragmentation
  - Relation is partitioned into several fragments stored in distinct sites
- Replication and fragmentation can be combined
  - Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.

Data Replication

- A relation or fragment of a relation is **replicated** if it is stored redundantly in two or more sites.
- **Full replication** of a relation is the case where the relation is stored at all sites.

Data Replication (Cont.)

- Advantages of Replication
  - **Availability**: failure of site containing relation *r* does not result in unavailability of *r* if replicas exist.
  - **Parallelism**: queries on *r* may be processed by several nodes in parallel.
  - **Reduced data transfer**: relation *r* is available locally at each site containing a replica of *r*.
- Disadvantages of Replication
  - Increased cost of updates: each replica of relation *r* must be updated.
  - Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
    - One solution: choose one copy as **primary copy** and apply concurrency control operations on primary copy

Data Fragmentation

- Division of relation r into fragments r_1, r_2, …, r_n which contain sufficient information to reconstruct relation r.
- **Horizontal fragmentation**: each tuple of *r* is assigned to one or more fragments
- **Vertical fragmentation**: the schema for relation *r* is split into several smaller schemas
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

Horizontal Fragmentation of *account* Relation

branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$$account_1 = \sigma_{branch\_name=\text{“Hillside”}}(account)$$

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$$account_2 = \sigma_{branch\_name=\text{“Valleyview”}}(account)$$

Vertical Fragmentation of *employee_info* Relation

branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}\,(employee\_info\,)$$

account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}\,(employee\_info\,)$$

Advantages of Fragmentation

- Horizontal:
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed
- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
  - allows parallel processing on a relation
- Vertical and horizontal fragmentation can be mixed.
  - Fragments may be successively fragmented to an arbitrary depth.

Data Transparency

- **Data transparency**: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system
- Consider transparency issues in relation to:
  - **Fragmentation transparency**
  - **Replication transparency**
  - **Location transparency**

Chapter 19: Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- **Distributed Transactions**
- Commit Protocols
- Concurrency Control in Distributed Databases
- Distributed Query Processing
- Heterogeneous Distributed Databases

Distributed Transactions

- Transaction may access data at several sites.
- Each site has a local **transaction manager** responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a **transaction coordinator**, which is responsible for:
  - Starting the execution of transactions that originate at the site.
  - Distributing subtransactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Chapter 19: Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- **Commit Protocols**
- Concurrency Control in Distributed Databases
- Distributed Query Processing
- Heterogeneous Distributed Databases

Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another
- The *two-phase commit* (2PC) protocol is widely used
- The *three-phase commit* (3PC) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol. This protocol is not used in practice.

Two Phase Commit Protocol (2PC)

- Assumes **fail-stop** model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.
- The protocol involves all the sites at which the transaction executed
- Let T be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

Phase 1: Obtaining a Decision

- Coordinator asks all participants to ***prepare*** to commit transaction *T*.
  - C_i adds the records <**prepare** *T*> to the log and forces log to stable storage
  - sends **prepare** *T* messages to all sites at which *T* executed
- Upon receiving message, transaction manager at each site determines if it can commit the transaction
  - if not, add a record <**no** *T*> to the log and send **abort** *T* message to C_i
  - if the transaction can be committed, then:
  - add the record <**ready** *T*> to the log
  - force *all records* for *T* to stable storage
  - send **ready** *T* message to C_i

Phase 2: Recording the Decision

- *T* can be committed of C_i received a **ready** *T* message from all the participating sites: otherwise *T* must be aborted.
- Coordinator adds a decision record, <**commit** *T*> or <a**bort** *T*>, to the log and forces record onto stable storage.
- Coordinator sends a message to each participant informing it of the decision (commit or abort)
- Participants take appropriate action locally.

Handling of Failures - Site Failure

When a site S_k recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contain <**commit** *T*> record: site executes **redo** (*T*)
- Log contains <**abort** *T*> record: site executes **undo** (*T*)
- Log contains <**ready** *T*> record: site must consult C_i to determine the fate of *T*.
  - If *T* committed, **redo** (*T*)
  - If *T* aborted, **undo** (*T*)
- The log contains no control records concerning *T* (i.e., S_k failed before responding to the **prepare** *T* message from C_i)
  - since the failure of S_k precludes the sending of such a response, C_1 must abort *T*
  - S_k must execute **undo** (*T*)

Chapter 19: Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- Commit Protocols
- **Concurrency Control in Distributed Databases**
- Distributed Query Processing
- Heterogeneous Distributed Databases

Concurrency Control

- Modify concurrency control schemes for use in distributed environment.
- We assume that each site participates in the execution of a commit protocol to ensure global transaction automicity.
- We assume all replicas of any item are updated
  - Will see how to relax this in case of site failures later

Single-Lock-Manager Approach

- System maintains a *single* lock manager that resides in a *single* chosen site, say S_i
- When a transaction needs to lock a data item, it sends a lock request to S_i and lock manager determines whether the lock can be granted immediately
  - If yes, lock manager sends a message to the site which initiated the request
  - If no, request is delayed until it can be granted, at which time a message is sent to the initiating site

Single-Lock-Manager Approach (Cont.)

- Advantages of scheme:
  - Simple implementation
  - Simple deadlock handling
- Disadvantages of scheme are:
  - Bottleneck: lock manager site becomes a bottleneck
  - Vulnerability: system is vulnerable to lock manager site failure.

Distributed Lock Manager

- In this approach, functionality of locking is implemented by lock managers at each site
  - Lock managers control access to local data items
    - But special protocols may be used for replicas
- Advantage: work is distributed and can be made robust to failures
- Disadvantage: deadlock detection is more complicated
  - Lock managers cooperate for deadlock detection
    - More on this later
- Several variants of this approach
  - Primary copy
  - Majority protocol
  - Biased protocol
  - Quorum consensus

Distributed Query Processing

- For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses.
- In a distributed system, other issues must be taken into account:
  - The cost of a data transmission over the network.
  - The potential gain in performance from having several sites process parts of the query in parallel.

Query Transformation

- Translating algebraic queries on fragments.
  - It must be possible to construct relation *r* from its fragments
  - Replace relation *r* by the expression to construct relation *r* from its fragments
- Consider the horizontal fragmentation of the *account* relation into

$$account_1 = \sigma_{\,branch\_name\, = \text{ “Hillside”}} (account)$$

$$account_2 = \sigma_{\,branch\_name\, = \text{ “Valleyview”}} (account)$$

- The query $\sigma_{\,branch\_name\, = \text{ “Hillside”}} (account)$ becomes

$$\sigma_{\,\text{branch\_name = “Hillside”}} (account_1 \cup account_2)$$

which is optimized into

$$\sigma_{\,branch\_name\, = \text{ “Hillside”}} (account_1) \cup \sigma_{\,branch\_name\, = \text{ “Hillside”}} (account_2)$$

Simple Join Processing

- Consider the following relational algebra expression in which the three relations are neither replicated nor fragmented

  account $\bowtie$ *depositor* $\bowtie$ *branch*

- *account* is stored at site S_1
- *depositor* at S_2
- *branch* at S_3
- For a query issued at site S_I, the system needs to produce the result at site S_I

Possible Query Processing Strategies

- Ship copies of all three relations to site S_I and choose a strategy for processing the entire locally at site $S_{I.}$
- Ship a copy of the account relation to site S_2 and compute $temp_1$ = *account* $\bowtie$ *depositor at* S_2. Ship $temp_1$ from S_2 to S_3, and compute $temp_2 = temp_1$ *branch* at S_3. Ship the result $temp_2$ to S_I.
- Devise similar strategies, exchanging the roles S_1, S_2, S_3
- Must consider following factors:
  - amount of data being shipped
  - cost of transmitting a data block between sites
  - relative processing speed at each site

Heterogeneous Distributed Databases

- Many database applications require data from a variety of preexisting databases located in a heterogeneous collection of hardware and software platforms
- Data models may differ (hierarchical, relational, etc.)
- Transaction commit protocols may be incompatible
- Concurrency control may be based on different techniques
- System-level details almost certainly are totally incompatible.
- A **multidatabase system** is a software layer on top of existing database systems, which is designed to manipulate information in heterogeneous databases
  - Creates an illusion of logical database integration without any physical database integration

Advantages

- Preservation of investment in existing
  - hardware
  - system software
  - Applications
- Local autonomy and administrative control
- Allows use of special-purpose DBMSs
- Step towards a unified homogeneous DBMS
  - Full integration into a homogeneous DBMS faces
    - Technical difficulties and cost of conversion
    - Organizational/political difficulties
      - Organizations do not want to give up control on their data
      - Local databases wish to retain a great deal of **autonomy**

Query Processing

- Several issues in query processing in a heterogeneous database
- Schema translation
  - Write a **wrapper** for each data source to translate data to a global schema
  - Wrappers must also translate updates on global schema to updates on local schema
- Limited query capabilities
  - Some data sources allow only restricted forms of selections
    - E.g., web forms, flat file data sources
  - Queries have to be broken up and processed partly at the source and partly at a different site
- Removal of duplicate information when sites have overlapping information
  - Decide which sites to execute query
- Global query optimization