Chapter 22: Distributed Databases

Database System Concepts, 5th Ed.

Chapter 22: Distributed Databases

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

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 compatible software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software
  - 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—schema mapping 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

Heterogeneous Distributed Databases

- Many database applications require data from a variety of preexisting databases located in a heterogeneous collection of hardware and software platforms
- A **middleware 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
- Schema translation
  - Write a **wrapper** for each data source to translate to the global schema
  - Wrappers must translate queries on global schema to on different local schemas and then convert and assemble local answers into a global one

Homogeneous Distributed Data Storage

- Assume relational data model and every site can refer to global schema
- 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.
- Fully redundant databases are those in which every site contains a copy of the entire database.

Data Replication (Cont.)

- Advantages of Replication
  - **Availability**: failure of site containing relation *r* does not result in unavailability of *r* is 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 $r_1, r_2, \ldots, r_n$ 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.
- Example : relation account with following schema
- *Account* = (*branch_name*, *account_number, balance*)

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

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

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

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.

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.

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

Semijoin Strategy

- Let r_1 be a relation with schema R_1 stores at site S_1

  Let r_2 be a relation with schema R_2 stores at site S_2
- To evaluate the expression $r_1 \bowtie r_2$ and obtain the result at S_1 do:
  1. Compute $temp_1 \leftarrow \prod_{R1 \cap R2}$ (r1) at $S1$.
  2. Ship $temp_1$ from S_1 to S_2.
  3. Compute $temp_2 \leftarrow r_2 \bowtie$ temp1 at S_2
  4. Ship $temp_2$ from S_2 to S_1.
  5. Compute $r_1 \bowtie temp_2$ at S_1. This is the same as $r_1 \bowtie r_2$.

Join Strategies that Exploit Parallelism

- Consider $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ where relation r_i is stored at site S_i. The result must be presented at site S_1.
- r_1 is shipped to S_2 and $r_1 \bowtie r_2$ is computed at S_2: simultaneously r_3 is shipped to S_4 and $r_3 \bowtie r_4$ is computed at S_4
- S_2 ships tuples of $(r_1 \bowtie r_2)$ to S_1 as they produced;
  S_4 ships tuples of $(r_3 \bowtie r_4)$ to S_1
- Once tuples of $(r_1 \bowtie r_2)$ and $(r_3 \bowtie r_4)$ arrive at S_1 $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$ is computed in parallel with the computation of $(r_1 \bowtie r_2)$ at S_2 and the computation of $(r_3 \bowtie r_4)$ at S_4.

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.

System Failure Modes

- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of messages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - **Network partition**
    - A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them
      - Note: a subsystem may consist of a single node
- Network partitioning and site failures are generally indistinguishable.

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.)

- The transaction can read the data item from *any* one of the sites at which a replica of the data item resides.
- Writes must be performed on all replicas of a data item
- 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

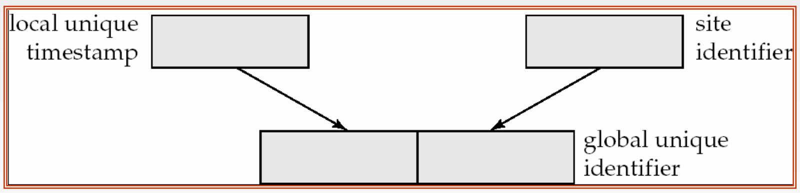
- In this approach, functionality of locking is implemented by lock managers at each site
  - Lock managers control access to local data items
    - But special protocols may be used for replicas
- Advantage: work is distributed and can be made robust to failures
- Disadvantage: deadlock detection is 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

Primary Copy

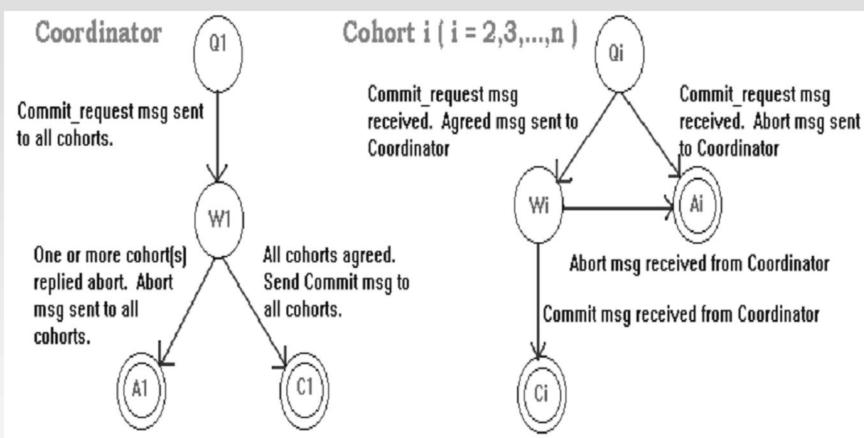
- Choose one replica of data item to be the **primary copy**.
  - Site containing the replica is called the **primary site** for that data item
  - Different data items can have different primary sites
- When a transaction needs to lock a data item *Q*, it requests a lock at the primary site of *Q*.
  - Implicitly gets lock on all replicas of the data item
- Benefit
  - Concurrency control for replicated data handled similarly to unreplicated data - simple implementation.
- Drawback
  - If the primary site of *Q* fails, *Q* is inaccessible even though other sites containing a replica may be accessible.

Timestamp-based Protocols

- Timestamp based concurrency-control protocols can be used in distributed systems
- Each transaction must be given a unique timestamp
- Main problem: how to generate a timestamp in a distributed fashion
  - Each site generates a unique local timestamp using either a logical counter or the local clock.
  - Global unique timestamp is obtained by concatenating the unique local timestamp with the unique identifier.



Finite State Diagram of Commit Protocol for Coordinator and Cohort



Q: Query state, W: Wait state, A: Abort state, C: Commit

The COORDINATOR:

Q1. The COORDINATOR sends the message to each COHORT. The COORDINATOR is now in the preparing transaction state.

W1. Now the COORDINATOR waits for responses from each of the COHORTS

- If any COHORT responds ABORT then the transaction must be aborted,
- After all COHORTS respond AGREED then the transaction is commited.
- If after some time period all COHORTS do not respond the COORDINATOR can send a COMMIT-REQUEST messages to the COHORTS that have not responded, or it can either transmit ABORT messages (and eventually it will do so if it does not get any answer)

Each Cohort (a.k.a. Participant)

The i-th **cohort** completes its local work (**Qi**), and decides whether it would like to commit or abort. Upon receiving the Commit_request from the coordinator, the cohort communicates its choice and

- If its decision is to commit it goes to wait state **Wi**.
- If its decision is to abort its goes to Abort state **Ai**

In **Wi** the cohort waits for the message from the coordinator.

- If the instruction from the coordinator is commit, then the cohort commits (state **Ci**)
- If the instruction from the coordinator is abort, then the cohort aborts (state **Ai)**
- **If the cohorts receives no instruction then the coordinator must wait holding on to all its resources: <u>blocking</u>** ☹

Failures

- Site Failure
- Coordinator Failure
- Communication Line Failure

Three Phase Commit (3PC)

- Avoids the blocking problem under the assumption that:
  - The is no network partitioning
  - <K sites fail (participants as well as coordinator)
- Initial Phase as 2PC
- When the coordinator reach commit decision it must first recording it in at least K sites (precommit phase) before it can proceed with (i) sending the actual decision to all sites and (ii) implementing it locally
- Knowledge of pre-commit decision can be used to commit despite coordinator failure
  - Avoids blocking problem as long as < K sites fail
- Drawbacks:
  - higher overheads
  - assumptions may not be satisfied in practice

End of Chapter

Database System Concepts, 5th Ed.