Database Development and Design (CPT201)

Lecture 11: Introduction to Distributed Databases

Dr. Wei Wang

Department of Computing

Learning Outcomes

- Distributed System Concepts
- Distributed Data Storage
- Distributed Transactions
- Distributed Query Processing
- Concurrency Control in Distributed Databases
- Failure Recovery in 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

Introduction

- Data is stored across several sites, each managed by a DBMS that can run independently.
- The location of data on each individual sites impacts query optimisation, concurrency control and recovery.
- Distributed data is governed by factors such as local ownership, increased availability, and performance issues.

Introduction cont'd

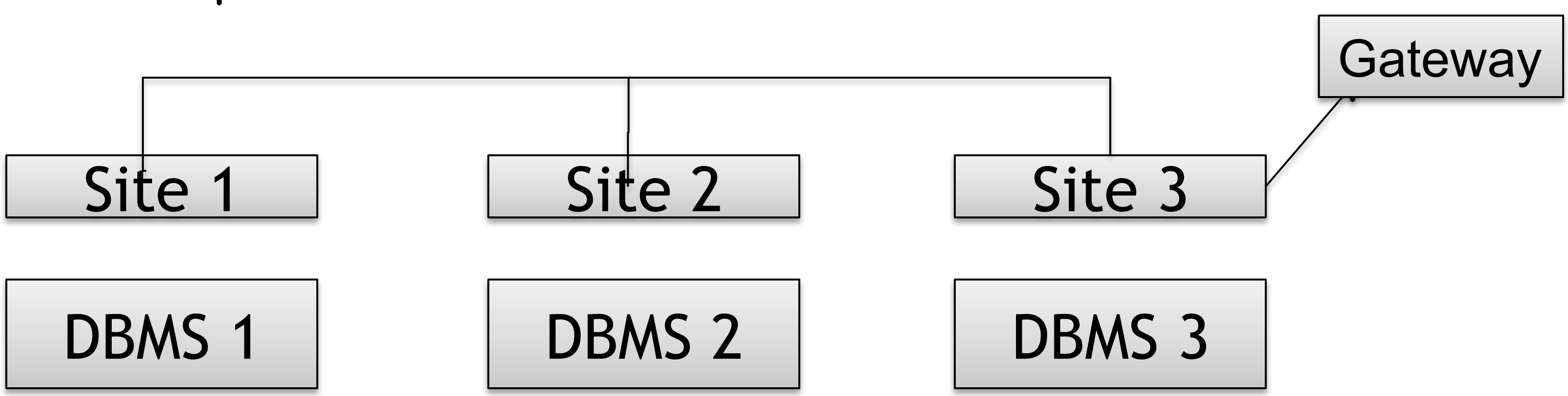
- Distributed Data Independence: Users should not have to know where data is located.
  - reference relations, copies or fragments of the relations.
  - extends Physical and Logical Data Independence principles
- Distributed Transaction Atomicity: Users should be able to write transactions that access and update data at several sites.
  - Transactions are atomic, all changes persist if the transaction commits, or rollback if transaction aborts.

Introduction cont'd

- If sites are connected by slow networks, these properties are hard to support efficiently.
- Users have to be aware of where data is located, i.e. Distributed Data Independence and Distributed Transaction Atomicity are not supported.
- For globally distributed sites, these properties may not even be desirable due to administrative overheads of making locations of data transparent.

Types of Distributed Databases

- **Homogeneous** – data is distributed but all servers run the same DBMS software.
- **Heterogeneous** – different sites run different DBMSs separately and are connected to enable access to data from multiple sites.
  - Gateway protocols - API that exposes DBMS functionality to external applications.
  - Examples: ODBC and JDBC

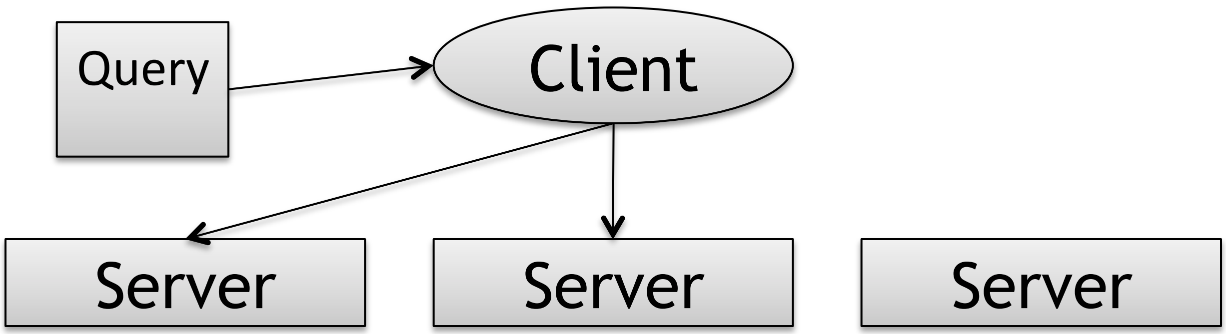


Architectures

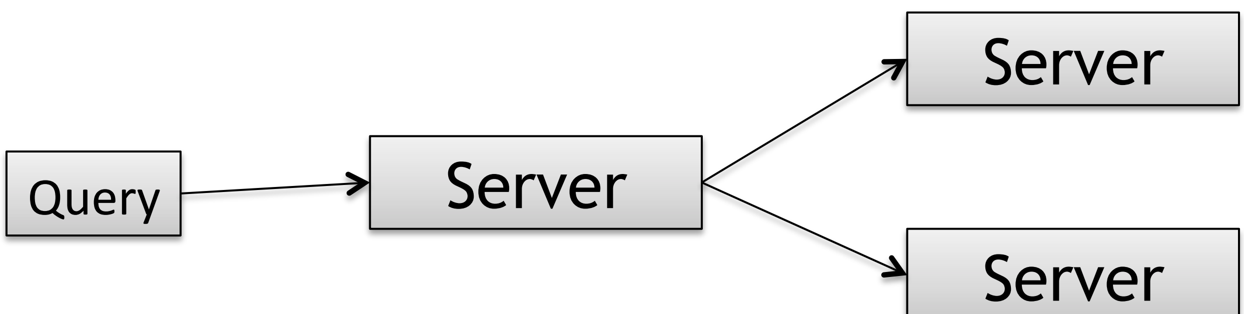
- **Client Server** – a system that has one or more client processes and one or more server processes. Client sends a query to a server, and the server processes the query returning the result to the client.
- **Collaborating Server** - capable of running queries against local data and executes transactions across multiple servers.
- **Middleware** – One database server can manage queries and transactions spanning across multiple servers. A layer that executes relational operations on data from other servers but does not maintain any data.

Architectures cont'd

Client-Server Architecture



Collaborated Server Architecture



Storing data

- Relations are stored across several sites. To reduce message-passing costs a relation maybe fragmented across sites.
- Fragmentation: breaks a relation to smaller relations and stores the fragments at different sites.
  - Horizontal fragments (HF) - rows of the original data.
    - Selection queries, fragments by city
    - **Disjoint union** of the HF must be equal to the original relation.
  - Vertical fragments (VF) - columns of the original data.
    - Projection queries, e.g., fragments of the first two columns
    - Collection of VF must be a **loss-less join decomposition**.

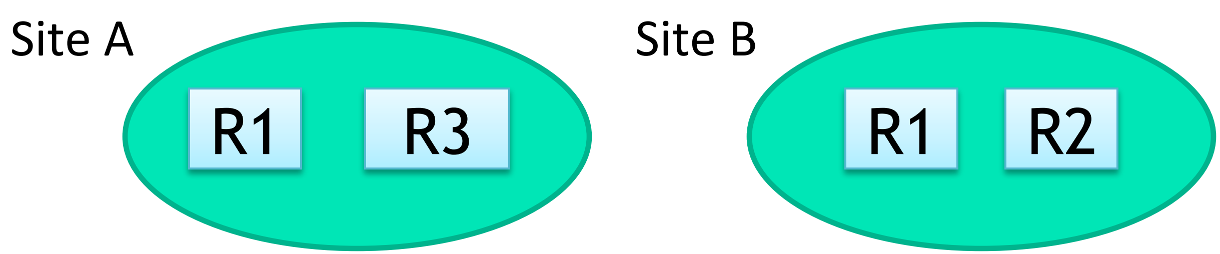
T1	Eid	Name	City
T2	123	Smith	Chicago
T3	124	Smith	Chicago
T4	125	Jones	Madras

HF

VF

Storing data cont'd

- Replication – storing several copies of a relation or fragment. Entire relation can be stored at one or more sites.
  - Increased Availability – If a site contains replicated data goes down, then we can use another site.
  - Faster Query Evaluation – Queries are executed faster by using local copy of a relation instead of going to a remote site.
  - Two kinds of replication are Synchronous and Asynchronous replication.



Distributed Catalog Management

- Must keep track of how data is distributed across sites.
- Must be able to give a unique identifier to each replica of each fragment/relation.
  - Global relation name- <local-name>, <birth-site>
  - Global replica name – *replica id* plus global relation name
- Site catalog - Describes all objects (fragments, replicas) at a site and keeps track of replicas of relations created at this site.
  - To find a relation look up its birth-site catalog
  - Birth-site never changes even if the relation is moved

Updating Distributed Data

- Users should be able to update data without worrying where relations are stored.
- **Synchronous replication** – all copies of a modified relation are updated before the modifying transaction commits.
- **Asynchronous replication** – copies of modified relation are updated over a period of time, and a transaction that reads different copies of the same relation may see different values.
  - Widely used in commercial distributed DBMSs
  - Users must be aware of distributed databases

Synchronous Replication

- **Voting technique** – a transaction must write a majority of copies to modify an object; read at least enough copies to make sure one of the copies is current.
  - For example, 10 copies, 7 are updatable, 4 are read
  - Each copy has a version number, the highest is the most current.
  - Not attractive and efficient, because reading an object requires reading several copies. Objects are read more than updated.
- **Read-any-write-all technique** – a transaction can read only one copy, but must write to all copies.
  - Reads are faster than writes especially if it's a local copy
  - Attractive when reads occur more than writes
  - Most common technique

Cost of Synchronous Replication

- **Read-any-write-all** cost - Before an update transaction can commit, it must lock all copies
  - Transaction sends lock requests to remote sites and waits for the locks to be granted, during a long period, it continues to hold all locks.
  - If there is a site or communication failure then transaction cannot commit until all sites are recovered
  - Committing creates several additional messages to be sent as part of a commit protocol
- Since synchronous replication is expensive, Asynchronous replication is gaining popularity even though different copies can have different values.

Asynchronous Replication

- Allows modifying transactions to commit before all copies have been changed.
  - Users must be aware of which copy they are reading, and that copy may be out-of-sync for short period of time.
- Two approaches: Primary Site and Peer-to-Peer replication.
  - Difference lies in how many copies are "updatable" or "master copies".

Peer to Peer Asynchronous Replication

- More than one copy can be designated as updateable (i.e. master copy).
- Changes to a master copy must be propagated to other copies somehow.
- Conflict resolution is used to deal with changes at different sites.
  - Each master is allowed to update only one fragment of the relation, and any two fragments updatable by different masters are disjoint.
  - Updating rights are held by one master at a time.

Primary Site Replication

- **Primary site** – one copy of a relation is the master copy
- **Secondary site**- replicas of the entire relation are created at other sites. They cannot be updated.
- Users register/publish a relation at the primary site and subscribe to a fragment of the relation at the secondary site.
- Changes to the primary copy transmitted to the secondary copies are done in two steps.
  - First capture changes made by committed transactions, then apply these changes.

Primary Site Asynchronous Replication - Capture

- **Log Based Capture** - the log maintained for recovery is used to generate a Change Data Table (CDT)
- **Procedural Capture** – A procedure that is invoked by the DBMS which takes a snapshot of the primary copy
- Log based capture is generally better because it deals with changes to the data and not the entire database. However it relies on log details which may be system specific.

Primary Site Asynchronous Replication - Apply

- The Apply process at the secondary site periodically obtains a snapshot of the primary copy or changes to the CDT table from the primary site, and updates the copy.
  - Period can be timer or user's application program based.
- Replica can be a view over the modified relation.
- Log-Based Capture plus continuous Apply minimises delay in propagating changes.
- Procedural Capture plus application-driven Apply is the most flexible way to process updates.
  - Used in data warehousing applications

Data Warehousing

- Creating giant warehouses of data from many sites
  - Create a copy of all the data at some locations
  - Use the copy rather than going to individual sources.
  - Enable complex decision support queries
- Seen as an instance of asynchronous replication; copies are updated infrequently.
- Source data controlled by different DBMSs
- Need to clean data and remove mismatches while creating replicas.

Distributed Queries

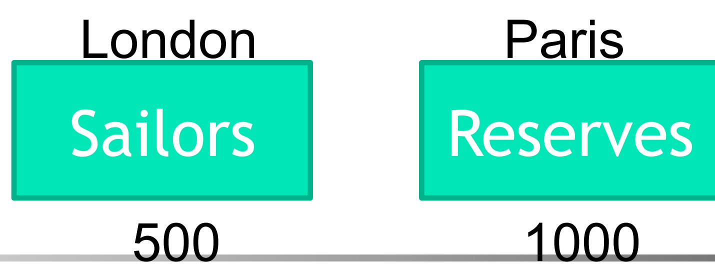
Example query with a relation S (fragmented at Shanghai and Tokyo sites):

```
SELECT AVG(S.age)
FROM Sailors S
WHERE S.rating > 3 AND S.rating < 7
```

- Horizontally Fragmented
  - Tuples with rating < 5 at Shanghai, >= 5 at Tokyo. When calculating average, must compute sum and count at both sites
  - If WHERE contained just S.rating > 6, just one site.
- Vertically Fragmented
  - *sid* and *rating* at Shanghai, *sname* and *age* at Tokyo, *tid* at both.
  - Joining two fragments by a common *tid* and execute the query over this reconstructed relation
- Replicated
  - Since relation is copied to more than one site, choose a site based on local cost.

Distributed Joins

- Joins of relations across different sites can be very expensive.
- Fetch as needed
  - For example: Sailors relation is stored at London and Reserves relation is stored in Paris. There are 500 blocks of Sailors and 1,000 blocks of Reserves
  - Use the block nested loops join in London with Sailors as the outer join, and for each Sailors block, fetch all Reserves blocks from Paris.
  - The cost is: 500 D + 500 * 1000 (D + S)
    - 500D = the time to scan Sailors
    - 500*1000 (D+S) = for each Sailor's block the cost of scanning and shipping all of Reserves
      - D is the cost of read/write blocks
      - S is the cost to ship a block
  - If query was not submitted at London, must add cost of shipping result to query site.
  - Can also do index nested loops join in London, fetching matching Reserves tuples for each Sailors tuple as needed.



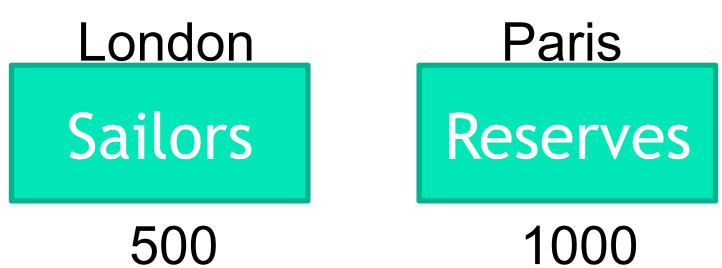
Distributed Joins cont'd

```
SELECT *
FROM Sailors S, Reserves R
WHERE S.sid = Reserves.sid
```

- **Fetch as Needed**, block nested loop join, Sailors as outer:
  - **Cost:** 500 D + 500 * 1000 (D+S)
  - **D** is cost to read/write page; **S** is cost to ship page.
  - S is very large, so cost is essentially 500,000S
- **Ship to One Site** – merge join, Ship Reserves to London.
  - Assume only one relation can be read in memory (sorting can be done in memory), cost = 3*(500+1000)
  - Cost: 1000S + 4500D, essentially 1000S as D<<S.

Semi Join & Bloom Join

- Assume shipping Reserves to London and computing the join at London, and some of those tuples in Reserves do not join with any tuples in Sailors.
  - Need to identify Reserve tuples that guarantee not to join with any Sailors tuples.
  - Two techniques: Semi Joins and Bloom Joins



Semi Join

- Three steps in reducing the number of Reserves tuples to be shipped.
  - At London, compute projection of Sailors onto the join attribute and ship this projection to Paris. (sids)
  - At Paris, join Sailors projection with Reserves. Ship the join result to London. This result is called 'reduction of Reserves with respect to Sailors'.
  - At London, compute the join of the reduction of Reserves with sailors.
- Idea: tradeoff the cost of computing and shipping projection for cost of shipping full Reserves relation.
  - Especially useful if there is a selection on Sailors, and answer desired at London.

Bloom Join

- Bloom Join is similar to Semi Join but there is a bit-vector shipped in the first step instead of a projection.
- At London, compute a bit-vector of some size k:
  - Hash each tuple of Sailors (using sid) into range 0 to k-1.
  - If some tuple hashes to i, set bit i to 1 ($0 \le i \le k-1$).
  - Ship bit-vector to Paris.
- At Paris, hash each tuple of Reserves (using sid) similarly, and discard tuples that are hashed to 0 in Sailors bit-vector (no Sailors tuples hash to the *i*th partition).
  - Result is called 'reduction of Reserves with respect to Sailors'.
- Ship bit-vector reduced Reserves to London.
- At London, join Sailors with reduced Reserves.
- Bit-vector cheaper to ship, almost always efficient.

Distributed Query Optimisation

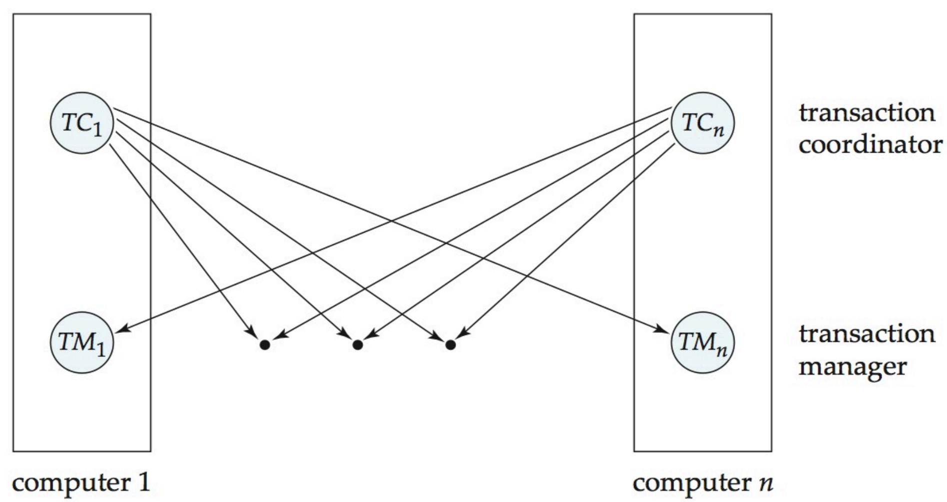
- Consider all plans, pick cheapest; similar to centralised optimisation.
  - Communication costs, if there are several copies of a relation, need to decide which to use.
  - If individual sites are running a different DBMS, autonomy of each local site must be preserved while doing global query planning.
  - Use new distributed join methods.
- Query site constructs a global plan, with suggested local plans describing processing at each site. If a site can improve suggested local plan, free to do so.

Distributed Transactions

- Transaction is submitted at one site but can access data at other sites.
- Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site, i.e.,
  - Maintaining a log for recovery purposes.
  - Participating in an appropriate concurrency-control scheme to coordinate the concurrent execution of the transactions executing at that site.
- Concurrency control and recovery
  - Assume Strict two-Phase Locking with deadlock detection is used
    - If a transaction wants to read an object it first requests a shared lock on the object.
    - All exclusive locks held by a transaction are released when the transaction is completed.

Transaction Coordinator

- Transaction coordinator is responsible for:
  - Starting the execution of the transaction.
  - Breaking the transaction into a number of sub-transactions and distributing these sub-transactions to the appropriate sites for execution.
  - Coordinating the termination of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.

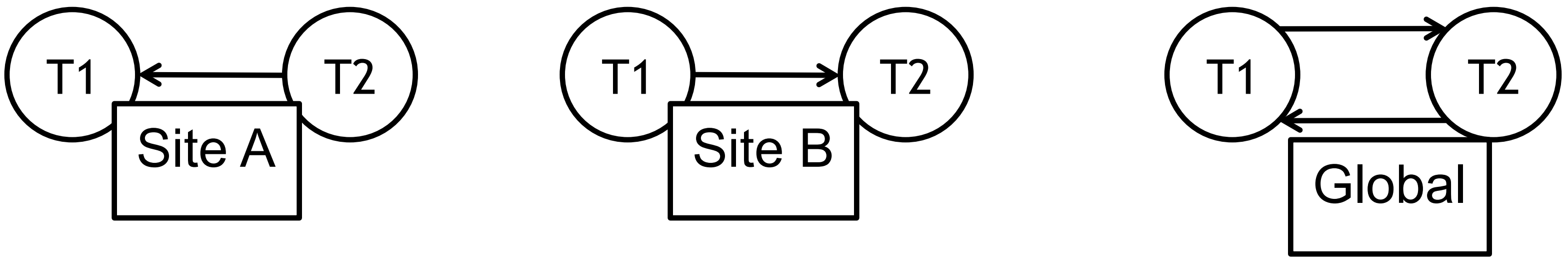


Distributed Locking Protocols

- When locks are obtained and released is determined by the concurrency control protocol.
- Lock management can be distributed across many sites:
  - Single-lock manager (Centralised) – One site does all the locking; vulnerable if one site goes down.
  - Primary Copy – Only one copy of each object is designated a primary copy, requests to lock/unlock are handled by lock manager at the primary site regardless where the copy is stored.
  - Distributed lock manager – Requests to lock/unlock a copy of an object stored at a site are handled by the lock manager at the site where the copy is stored.

Distributed Deadlock

- Each site maintains local waits-for graph, and a cycle in a local graph indicates a **deadlock**.
- A global deadlock might exist even if the local graphs contain no cycles



- Three algorithms of deadlock detection
  - Centralised – send all local graphs to one site that is responsible for deadlock detection.
  - Hierarchal – organise sites into a hierarchy and send local graphs to parent in the hierarchy.
  - Timeout – abort transaction if it waits too long

Distributed Recovery and Concurrency Control

- Recovery in distributed DBMSs is more complicated than in centralised DBMSs
  - e.g., failure of communication links; failure of a remote site at which a sub transaction is executing.
  - All of the sub transactions must commit or not commit at all. This must be guaranteed despite link failures.
  - Need a commit protocol – the most common one is Two-Phase Commit.
- A log is maintained at each site, as in a centralised DBMS, and commit protocol actions are additionally logged.

Commit Protocols: Two Phase Commit (2PC)

- The site at which the transaction originated is the coordinator. Other sites at which sub transactions are executed are subordinates.
- When a user decides to commit a transaction, the commit command is sent to the coordinator for the transaction. This initiates the **2PC**:
  - Phase 1: Coordinator sends a prepare message to each subordinate; When subordinate receives a prepare message, it decides to abort or commit its sub transaction; Subordinate forces writes an abort or ready log record and sends a no or yes message to coordinator accordingly.
  - Phase 2: If coordinator receives a yes message from all subordinates, force-writes a commit log record and sends commit message to all subordinates. Else force-writes a an abort log record and sends an abort message; Subordinates force-write abort or commit log record based on the message they receive.
  - In some implementations, after the two phases, subordinates send acknowledgement message to coordinator; After coordinator receives ack messages from all subordinates it writes an end log for the transaction.

READ TEXTBOOK

Recover After Failure

- When a site comes back from a crash there is a recovery process that reads the log and processes all transactions which executed the commit protocol at the time of the crash.
- Failure of a participating site
  - Examine its own logs, if there is commit or abort log record for transaction T, then redo/undo T respectively.
  - The log contains a <ready T> record, repeatedly contact the coordinator or other active sites to find the status of T, then performs redo/undo accordingly and write commit/abort log records depending on coordinator's response;
  - The log contains no control records (abort, commit, ready) concerning T, undo.
- Failure of the coordinator
  - participating sites must decide the fate of T.
  - If an active site contains a <commit T>/<abort T> record in its log, then T must be committed/aborted.
  - If some active site does not contain a <ready T> , preferable to abort T.
  - If active sites have a <ready T> record in their logs, but no additional control records (such as <abort T> or <commit T>), it is impossible to determine if a decision has been made, and what that decision is, until the coordinator recovers. (***in-doubt transaction***)

Blocking – Coordinator Failure

- If locking is used, an in-doubt transaction T may hold locks on data at active sites. Such a situation is undesirable, because it may be hours or days before coordinator is again active. This situation is called the blocking problem, because T is blocked pending the recovery of coordinator site.
- Solution:
  - Use **<ready T, L>** log record, where **L** is a list of all write locks held by the transaction T when the log record is written. At recovery time, after performing local recovery actions, for every in-doubt transaction T, all the write locks noted in the <ready T, L> log record (read from the log) are reacquired.
  - Can also be solved using the 3-phase commit protocol (under certain situations)

Recovery from Network Partitions

- The coordinator and all its participants remain in one partition. In this case, the failure has no effect on the commit protocol.
- The coordinator and its participants belong to several partitions. From the viewpoint of the sites in one of the partitions, it appears that the sites in other partitions have failed.
  - Sites that are not in the partition containing the coordinator simply execute the protocol to deal with failure of the coordinator.
  - The coordinator and the sites that are in the same partition: the coordinator follow the usual commit protocol, assuming that the sites in the other partitions have failed.

End of Lecture

- Summary
  - Distributed System Concepts
  - Distributed Data Storage
  - Distributed Transactions
  - Distributed Query Processing
  - Concurrency Control in Distributed Databases
  - Failure Recovery in Distributed Databases
- Reading
  - Textbook 6th edition, chapters 17.4, 17.5, 19.1, 19.2, 19.3, 19.4, 19.5
  - Textbook 7th edition, chapters 20.5, 21.2, 21.4, 22.9, 23.1, 23.2, 23.3